JPA 10-269353

(19) JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: **10269353 A**

(43) Date of publication of application: **09.10.98**

(51) Int. Cl **G06T 5/00**
G06T 1/00

(21) Application number: **10000359**

(22) Date of filing: **05.01.98**

(30) Priority: **02.01.97 US 97 35733**
27.10.97 US 97 957833

(71) Applicant: **RUNAWAY TECHNOL INC**

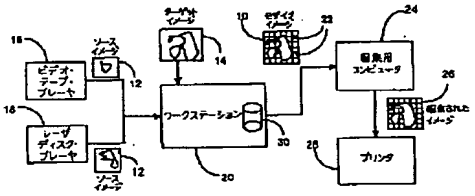
(72) Inventor: **SILVERS ROBERT S**

(54) **DIGITAL COMPOSITION OF MOSAIC IMAGE**

(57) Abstract:

PROBLEM TO BE SOLVED: To generate an artistically excellent mosaic image by executing a mosaic software and generating the mosaic image from a target image and a source image.

SOLUTION: The mosaic image 10 is generated from the picked-up source image 12 so as to permit it to approximate to the target image 14. A computer work station 20 having a video input is used in order to pick-up the source image from a video tape player 16 and a laser disk player 18. A mosaic program reads the source image 12 from the selection of a designated database, analyzes the target image and selects the source image 12 to be used in the respective tiles of the mosaic image 10. That is, the source image 12 provided with resolution corresponding to the number of sub-areas which are selected as against the mosaic image is loaded into a list where structure is linked.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平10－269353

(43)公開日　平成10年(1998)10月９日

(51)Int.Cl.[6]	識別記号	ＦＩ		
G06T 5/00		G06F 15/68		
1/00		15/66	470	J

審査請求　有　請求項の数30　ＯＬ　外国語出願　（全44頁）

(21)出願番号　特願平10－359

(22)出願日　平成10年(1998)１月５日

(31)優先権主張番号　６０／０３５７３３

(32)優先日　1997年１月２日

(33)優先権主張国　米国（ＵＳ）

(31)優先権主張番号　０８／９５７８３３

(32)優先日　1997年10月27日

(33)優先権主張国　米国（ＵＳ）

(72)発明者　ロバート・エス・シルバーズ
アメリカ合衆国マサチューセッツ州02139，ケンブリッジ，フランクリン・ストリート　129，ナンバー　105

(74)代理人　弁理士　社本　一夫　（外５名）

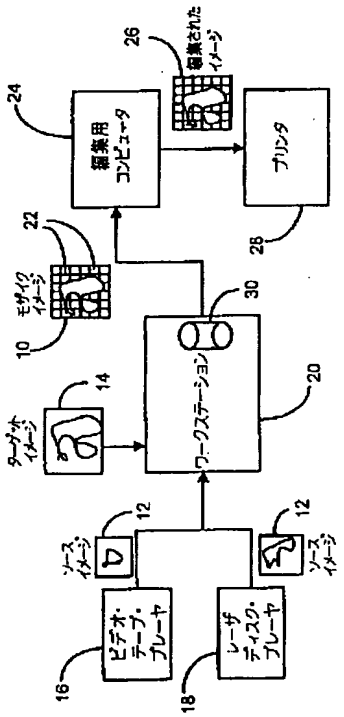
最終頁に続く

(54)【発明の名称】モザイク・イメージのデジタル・コンポジション

(57)【要約】

【課題】　ソース・イメージのデータベースからのモザイク・イメージの形成。

【解決手段】　モザイク・イメージを生成するために、ソース・イメージが分析され、選択され、編成される。ターゲット・イメージをタイル領域に分割し、自乗平均誤差の平方根を計算することにより、そのそれぞれを個々のソース・イメージの部分と比較し、得られる最も整合性の高いソース・イメージを決定する。それぞれの最もの整合性の高いソース・イメージを各タイル領域に置くことにより、モザイク・イメージを形成する。



1

【特許請求の範囲】

【請求項１】　複数のソース・イメージとコンピュータとを用いることにより、ターゲット・イメージに近似する外観を持つモザイク・イメージを生成する方法において、

ターゲット・イメージをコンピュータへロードするステップと、

前記ターゲット・イメージを、各々が該ターゲット・イメージの個別の位置を表す複数のタイル領域へ分割するステップと、を備え、

各タイル領域ごとに、

視覚的類似性の値を生成するためにソース・イメージをタイル領域と比較する比較ステップであって、各ソース・イメージの複数の部分を分析するステップを含む、比較ステップと、

タイル領域を表すために視覚的類似性の最大の値を持つソース・イメージを選択する選択ステップと、

選択された前記ソース・イメージを、前記モザイク・イメージにおいて、前記タイル領域の位置に対応する位置に配置するステップとを備える方法。

【請求項２】　請求項１記載の方法において、前記タイル領域を、各々が前記ソース・イメージの特定の部分に対応する個々のサブ領域に分割するステップと、視覚的類似性の値を生成するために個々のサブ領域を個々のソース・イメージ領域部分と比較するステップとを更に含む、方法。

【請求項３】　請求項２記載の方法において、各サブ領域当たり１つのピクセルを有するソース・イメージを用いるステップを更に含む、方法。

【請求項４】　請求項１記載の方法において、前記比較ステップは、赤、緑および青のチャネルの平均根二乗平均誤差を計算するステップを更に含む、方法。

【請求項５】　請求項１記載の方法において、１つのソース・イメージがモザイク・イメージにおいて１回より多く現れることのないように、前記選択ステップにおいて選択されたソース・イメージを考慮から除くステップを更に含む、方法。

【請求項６】　請求項１記載の方法において、ソース・イメージを捕捉する捕捉ステップと、捕捉されたソース・イメージをデータベースに記憶するステップとを更に含む、方法。

【請求項７】　請求項６記載の方法において、前記捕捉ステップにおいて捕捉されたソース・イメージを方形に切り取ることにより、変更されたソース・イメージを生成するステップを含む、方法。

【請求項８】　請求項７記載の方法において、ランドスケープ・フォーマットの捕捉されたソース・イメージの場合に、該捕捉されたイメージを中心から切取るステップを更に含む、方法。

【請求項９】　請求項８記載の方法において、ポートレート・フォーマットの捕捉されたソース・イメージの場合に、該捕捉されたイメージを中心より上から切り取るステップを更に含む、方法。

2

【請求項１０】　請求項７記載の方法において、データベース内の捕捉されたソース・イメージを分類するステップを更に含む、方法。

【請求項１１】　請求項７記載の方法において、捕捉されたソース・イメージを異なるレベルの解像度で記憶するステップを更に含む、方法。

【請求項１２】　請求項１記載の方法において、視覚的類似性の最大の値を持つ前記ソース・イメージを、もし前記ソース・イメージが別のタイル領域に対して視覚的類似性のより高い値を持つと判定されるならば、除外するステップを更に含む、方法。

【請求項１３】　請求項１記載の方法において、モザイク・イメージにおける確実な包含のための少なくとも１つのソース・イメージを指定するステップであって、確実にされたソース・イメージは、モザイク・イメージにおいて、視覚的類似性の最大の値を有するタイル領域の位置に対応する位置に配置される、ステップを更に含む、方法。

【請求項１４】　請求項１記載の方法において、前記ターゲット・イメージの予め定めた部分と排他的に整合するためにソース・イメージのサブカテゴリを指定するステップを更に含む、方法。

【請求項１５】　複数のソース・イメージを用いることによりターゲット・イメージに近似する外観を持つモザイク・イメージを生成する装置において、

ターゲット・イメージを、各々が前記ターゲット・イメージの個々の位置を表す複数のタイル領域に分割するように動作するモザイク生成ソフトウエアを実行するコンピュータ・ワークステーションを備え、

前記モザイク生成ソフトウエアは各タイル領域に処理を行うものであり、

複数のソース・イメージの部分を前記タイル領域と比較して視覚的類似性の値を生成し、

前記タイル領域を表すために、視覚的類似性の最大の値を持つソース・イメージを選択し、

選択された前記ソース・イメージを、前記モザイク・イメージにおいて、前記タイル領域の位置に対応する位置に配置するように動作する、

装置。

【請求項１６】　請求項１５記載の装置において、前記モザイク生成ソフトウエアは、前記タイル領域を個々のサブ領域に分割するように更に動作し、各サブ領域は前記ソース・イメージの特定の部分に対応し、各サブ領域は、視覚的類似性の値を生成するため各個のソース・イメージ部分と比較される、装置。

【請求項１７】　請求項１６記載の装置において、前記タイル領域との比較に用いられる前記ソース・イメージ

は各サブ領域当たり１つのピクセルを有する、装置。

【請求項１８】　請求項１５記載の装置において、前記モザイク生成ソフトウエアは、赤、緑および青のチャネルの平均根二乗平均誤差を計算するように更に動作する、装置。

【請求項１９】　請求項１５記載の装置において、前記モザイク生成ソフトウエアは、１つのソース・イメージがモザイク・イメージにおいて１回より多く現れないように、選択された被選択ソース・イメージを考慮から除外するように更に動作する、装置。

【請求項２０】　請求項１５記載の装置において、ビデオ・テープ・プレーヤとビデオディスク・プレーヤとからなるグループから選択されたビデオ装置を更に含み、該ビデオ装置は、コンピュータ・ワークステーションにおけるデータベースに記憶されるソース・イメージを捕捉するよう動作する、装置。

【請求項２１】　請求項２０記載の装置において、変更されたソース・イメージが、捕捉された前記ソース・イメージを一致したサイズにするように切り取り且つサイズを直すことにより生成される、装置。

【請求項２２】　請求項２１記載の装置において、ランスケープ・フォーマットの捕捉されたソース・イメージの場合に、捕捉された前記イメージは中心から切取られる、装置。

【請求項２３】　請求項２２記載の装置において、ポートレート・フォーマットの捕捉されたソース・イメージの場合に、捕捉された前記イメージは中心より上から切り取られる、装置。

【請求項２４】　請求項２１記載の装置において、捕捉された前記ソース・イメージはデータベース内で分類される、装置。

【請求項２５】　請求項２１記載の装置において、捕捉された前記ソース・イメージは、異なるレベルの解像度で記憶される、装置。

【請求項２６】　請求項２０記載の装置において、前記モザイク・イメージを編集するためのソフトウエアを有する編集用コンピュータを更に含む装置。

【請求項２７】　請求項２６記載の装置において、編集されたモザイク・イメージを印刷するプリンタを更に含む装置。

【請求項２８】　請求項１５記載の装置において、視覚的類似性の最大の値を持つ前記ソース・イメージが、もし前記ソース・イメージが別のタイル領域に対して視覚的類似性のより高い値を持つと判定されるならば、除外される、装置。

【請求項２９】　請求項１５記載の装置において、少なくとも１つのソース・イメージがモザイク・イメージにおいて確実に包含され、確実にされたソース・イメージは、モザイク・イメージにおいて、視覚的類似性の最大の値を有するタイル領域の位置に対応する位置に配置される、装置。

【請求項３０】　請求項１５記載の装置において、前記ターゲット・イメージの予め定めた部分と排他的に整合するように、ソース・イメージのサブカテゴリが指定される、装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、イメージのコンピュータ処理に関し、特に複数のサブイメージからの１つのイメージの生成に関する。

【０００２】

【従来の技術】コンピュータを用いるイメージの分析および処理は周知である。例えば、コンピュータは、異なる種類のコインを弁別してコインの合計値を計算するために、コンベア・ベルトで移動するコインのイメージを分析するのに用いられてきた。同様に、コンピュータは、製造中に欠陥を検出するために、集積回路および印刷回路板のイメージを分析するのに用いられてきた。特別な効果を生じるための写真の静止像およびフルモーションのビデオ・イメージの処理もまた周知である。しかし、これら公知の技法は、美術的に良いモザイク・イメージを生成するものではない。

【０００３】

【発明が解決しようとする課題】本発明によればターゲット・イメージのタイル部分を分析し、予め定めた基準に従って最良の一致を生じるためにデータベースからのソース・イメージとターゲット・イメージの分析された各タイル部分を比較し、ターゲット・イメージの分析された各タイル部分に対応するモザイク・イメージの各タイル部分に配置されたそれぞれの最良に一致するソース・イメージを備えるモザイク・イメージを生成することにより、ターゲット・イメージに近似するモザイク・イメージがソース・イメージのデータベースから生成される。一つの実施形態において、最良一致のための基準が、赤、緑および青（ＲＧＢ）の自乗平均誤差の平方根（Root-Mean-Square (RMS) error）のバージョンを計算することを含む。検討中のターゲット・イメージの領域と視覚的に最も類似するソース・イメージを見つけ出すことができる限り、他の一致システムを用いることもできる。

【０００４】サブ領域分析によりモザイク・イメージにおいて向上した解像度が実現される。特に、ターゲット・イメージにおける各タイル部分はサブ領域へ分割され、本例では、ＲＧＢのＲＭＳエラー（RMS error）分析を用いて、サブ領域を各ソース・イメージの対応するサブ領域と独立的に比較する。各サブ領域に対する計算されたＲＧＢのＲＭＳエラーが加算されて、ソース・イメージ全体に対する合計ＲＧＢ　ＲＭＳエラーを提供する。最小の合計ＲＧＢ　ＲＭＳエラーを持つ割り当てられていないイメージが、モザイク・イメージにおける対

5

応するタイル部分に使用されるように次に割り当てられる。サブ領域の使用は、詳細ではない（ディテールがない）領域にでさえ利益的であり、高周波ディテールの少ない領域に対して低いコントラストのイメージを選択することにより、色の更に均一な分布を生じる結果となる。別の実施の形態は、別の場所において低いエラーを有するであろうならば、ソース・イメージがモザイクの所与の場所に置かれることを阻止するための第２のパスを用いる。

【０００５】

【発明の実施の形態】図１は、ターゲット・イメージ１４に近似するように、捕捉されたソース・イメージ１２からモザイク・イメージ１０を生成する装置を示す。開示される実施の形態において、ビデオ・テープからのソース・イメージの捕捉を容易にするため、ＶＨＳビデオ・テープ・プレーヤ１６が用いられる。ビデオ・テープ・プレーヤは、ソース・イメージとして用いられる静止イメージを捕捉するためにビデオ・テープを通して単一ステップを行うために用いられる。或いはまた、ソース・イメージは、ビデオ・テープの再生中にリアルタイムで捕捉することができる。コンピュータで制御可能なレーザディスク・プレーヤ１８もまた、ソース・イメージの捕捉を容易にするため用いることができる。所望の主題のものが両方のソースから入手可能である時にはレーザディスクが好適である。なぜなら、レーザデイスクは品質がより高く、かつレーザ・ディスクから得られる静止イメージに対して容易にランダム・アクセスできるからである。開示された実施の形態においては、ビデオテープ・プレーヤ１６とレーザディスク・プレーヤ１８からソース・イメージ１２を捕捉するため、ビデオ入力を持つコンピュータ・ワークステーション２０が用いられる。コンピュータ・ワークステーション２０はまた、入力としてターゲット・イメージ１４を受け取り、モザイク・ソフトウエアを実行することによりターゲット・イメージとソース・イメージとからモザイク・イメージ１０を生成するために用いられる。モザイク・ソフトウエアにより生成されるモザイク・イメージ１０はタイル２２のアレイを含み、ここで、各タイル２２がソース・イメージ１２であり、モザイク・イメージ１０の外観全体がターゲット・イメージ１４の外観に近似する。編集されたモザイク・イメージ２６を生成するために、アドーブ・フォトショップ（Adobe Photoshop）（登録商標）のようなイメージ編集ソフトウエアを備えたマッキントッシュ（登録商標）、ＰＣまたはＵＮＩＸ（登録商標）に基くシステムのような編集コンピュータ２４を、モザイク・イメージ２２を編集するために用いることができる。編集されたモザイク・イメージ２６を印刷するためにプリンタ出力装置２８を用いることができる。

【０００６】捕捉されたソース・イメージ１２は、分析され、ワークステーション２０に維持されるデータベース３０に記憶されることが可能である。生の捕捉したソース・イメージ１２を分析してこれから新たなソース・イメージを生成するために、ａｄｄ＿ｉｍａｇｅｓ＿ｔｏ＿ｄａｔａｂａｓｅ（イメージをデータベースに付加）プログラムが用いられる。より詳細には、ａｄｄ＿ｉｍａｇｅｓ＿ｔｏ＿ｄａｔａｂａｓｅプログラムは、ファイルシステムのディレクトリのリストと、イメージ・サイズと、出力パスとを入力として受け取り、応答して指定された各ディレクトリを開いてソース・イメージをサーチするよう動作し、このソース・イメージから指定された寸法に切り取ってサイズを直す。その後、正方形（square）が出力経路により指定される場所へ移動される。一実施形態においては、ソース・イメージがランドスケープ・フォーマットであるならば、正方形イメージがソース・イメージの中心から切り取られる。ソース・イメージがポートレート・フォーマットであるならば、正方形がソース・イメージの中心と上部との間から切り取られる。その結果、正方形イメージは、エッジを切落すことなく、人の顔の如きソース・イメージの強調された特徴を含むであろうようになる。次に、イメージがデータベース３０に格納される。データベース３０は、主題とサイズにより分類されるディレクトリにフォーマット化されたイメージを保持するファイル・システムである。

【０００７】図２は、データベース３０（図１）内のソース・イメージ１２の編成を示す。ソース・イメージ１２は、動物ルート・ノード３２、人ルート・ノード３４、場所ルート・ノード３６の如きルート・ノード（root node）の下に分類されて配置される。動物のソース・イメージからモザイク・イメージを生成するためには、動物ルート・ノード３２がモザイク・ソフトウエアに対して選択される。動物ルート・ノード３２の下にあるディレクトリは、異なる解像度レベルの同じイメージ・ファイルを含むサブディレクトリである。オリジナル・サブディレクトリ３８は、フルサイズでの各ソース・イメージ４０の切り取られていないバージョンを含む。ａｄｄ＿ｉｍａｇｅｓ＿ｔｏ＿ｄａｔａｂａｓｅプログラムからの結果が受け入れられない場合には、モザイク生成中にソース・イメージが再び切り取られるので、オリジナル・サブディレクトリ３８が維持される。２５６×２５６（ピクセル）４２および６４×６４（ピクセル）４４で示されるディレクトリは、フォーマット化されたソース・イメージの大きなバージョンを含み、それは、最終的なビット・マップを出力するために主として用いられる。本例においては、３２×３２（ピクセル）４６のディレクトリは、構築プロセスの間にスクリーン上でモザイク・イメージを見るために使用されるソース・イメージを含む。１６×１６（ピクセル）４８、８×８（ピクセル）５０、および１×１（ピクセル）５２のサブディレクトリは、モザイク・ソフトウエアが初期設

7

定される時にプレロード（preload）されるソース・イメージを含む。１６×１６（ピクセル）、８×８（ピクセル）および１×１（ピクセル）のサブディレクトリにおけるソース・イメージは、モザイク・イメージの生成中にソース・イメージをターゲット・イメージに整合させる（突き合わせる、matching）ために用いられる。他の解像度レベルのソース・イメージのディレクトリもまた維持される。

【０００８】図３は、モザイク・イメージを生成するための方法を示す。図２、図３および図４を参照する。ステップ６０に示されるように、ターゲット・イメージが選択されロードされる。次に、ステップ６２に示されるように、データベースにおけるソース・イメージのルート・ノードが選択されロードされる。より詳細には、データベースのパス（経路）が指定され、モザイク・プログラムが実行される。モザイク・プログラムは、ソース・イメージを、指定されたデータベース・パスにより示されるデータベースのセクションから読み出し、ターゲット・イメージを分析し、モザイク・イメージの各タイルで使用するソース・イメージを選択する。詳細には、モザイク・イメージに対して選択されたサブ領域の数（サブ領域の解像度）に対応する解像度を持つソース・イメージが、構造のリンクされたリストへロードされる。

8

【０００９】

【表１】

```
struct an_image {
    char *path;                          / データベースにおけるファイルのパスネーム /
    char used;                           /  イメージが使用されたかどうか /
    unsigned short *r;                   /RMS突き合わせのためのRGBイメージデータ/
    unsigned short *g;
    unsigned short *b;
    struct an_image *next;               / 次の構造へのポインタ /
    struct an_image *previous;           / 前の構造へのポインタ /
} an_image;
```

【００１０】例えば、モザイク・イメージにおける各タイルが８Ｘ軸サブ領域×８Ｙ軸サブ領域を含むならば、８×８（ピクセル）イメージがデータベースからロードされる。各軸に沿ったピクセルにおけるターゲット・イメージのサイズは、突き合わせ（matching）プロセス中に考察される所望のサブ領域の数で乗じられる出力タイル数に等しく、即ち、各軸に沿ってサブ領域当たり１ピクセルである。モザイク・イメージとターゲット・イメージの両方のＸ軸およびＹ軸に対して用いられる各タイル数が、ステップ６４に示されるように指定される。

【００１１】モザイク・プログラムは、ソース・イメージとターゲット・イメージとがいったんロードされると、突き合わせ（整合）プロセスを実行する。この突き合わせプロセスが開始すると、ターゲット・イメージは「ｘ」×「ｙ」タイル２２に分割される。ここで、（ｘ，ｙ）は、以下のようである。

【００１２】（target_image_width / width_subsamples, target_image_height / height_subsamples)　((ターゲットイメージ幅 ／ 幅サブサンプル, ターゲットイメージ高さ ／ 高さサブサンプル))

【００１３】ステップ６８に示されるように、新たなタイルがロードされる。次に、ステップ７０に示されるように、新たなサブ領域６６がロードされる。ローディングは、タイル２２の左上のサブ領域６６で開始し、各行を左から右へ移動し、上部から下部へ行ごとに移動する。次に、ステップ７２に示されるように、ロードされたサブ領域に対応するソース・イメージ・ピクセルがロードされる。

【００１４】突き合わせプロセスは、タイル２２をシリアル・ベースで個々に分析する。開示された実施の形態における各タイル２２について、各サブ領域６６の赤、緑および青（ＲＧＢ）のチャネルの平均根二乗平均誤差（average Root-Mean Square(RMS) error）の変動は、適切な解像度でありかつ「使用（used）」と表示されないデータベースにおける各ソース・イメージに対して、対応するソース・イメージ・ピクセルのそれぞれと比較される。ロードされたピクセルとロードされたサブ領域との間のＲＭＳエラーは、ＲＧＢチャネルに対して計算され、ステップ７４に示されるように、タイルに対する現在の合計（running sum）として保持される。ステップ７６に示されるように、分析されていないサブ領域がタイルに存在するならば、流れはステップ７０へ戻る。全てのサブ領域が分析されたとステップ７６で判定されたならば、現在の合計のＲＧＢ　ＲＭＳエラーが、ステップ７８に示されるように、ソース・イメージとタイルに対して今までに計算された最小のエラーと比較される。このエラー合計がタイルに対する前に記録された何れのエラー合計よりも小さければ、そのエラー合計の値

とソース・イメージに対するインデックスとが、ステップ８０に示されるように、記録される。

【００１５】ソース・イメージの全てがタイルの類似性について分析された時、最小の計算されたＲＧＢ　ＲＭＳエラーを持つソース・イメージが、ターゲット・イメージにおけるタイルに対応するモザイク・イメージにおけるタイルに、即ちイメージにおける同じ場所におけるタイルに、割り当てられる。特定的には、ステップ８２において、データベースにおける他のソース・イメージがタイルと比較されていないと判定されたならば、ステップ８４において、新たなソース・イメージがロードされ、流れはステップ７０へ戻る。ステップ８２において、全てのソース・イメージがタイルと比較されたと判定されたならば、ステップ８６に示すように、最小の合計のエラーを持つソース・イメージが、当該タイルに割り当てられ、「使用」とマークされる。割り当てられたソース・イメージは、ソース・イメージがモザイク・イメージに１回より多く現れないように、「使用」とマークされる。

【００１６】ターゲット・イメージにおける各タイルに対して、突き合わせプロセスが反復される。完了すると、ソース・イメージのリストがテキスト・ファイルに書き込まれ、このファイルは、ソース・イメージの全解像度バージョンからビット・マップを構築するために最終のレンダリング・プログラムにより使用される。より特定的には、ステップ８８で、全てのタイルが検査されたと判定されたならば、ステップ９０において示されるように、各タイルに対しての最小の合計のエラーのソース・イメージのリストがテキスト・ファイルへ書き込まれ、ステップ９２に示されるように、モザイク・プログラムがこのリストを読み出してビット・マップをアセンブルする（組み立てる）。ステップ８８で、検査されていないタイルがまだ存在することが判定されたならば、流れはステップ６８へ戻る。

【００１７】ＲＭＳエラーの計算を含む突き合わせプロセスの１つのバリエーションは、下記のように実現される。（このプログラムは、オンラインの特許出願の方式に従った１つのイメージ入力範囲内に収まらないので、３つの表（表２、表３、表４）に分割した。）

【００１８】

【表２】

```
/*      このルーチンの目的は、どのソース写真が、ターゲット・イ
    メージの所                */
/* 与の領域（グリッド・スペース）と、最も視覚的に類似するかを見つけ
 ることである。 */

int find_matches(int x, int y)
{
     register i, rt, gt, bt;
     int low, result, ii, the tile;
     char imagename[256], best_path[256];
     unsigned short rmas[XMAX*YMAX], gmas[XMAX*YMAX], bmas
[XMAX*YMAX];

     the_tile = x+(y*sizex);

     /* ターゲット・イメージのこの所与のグリッド位置に対して、エ
 ラーのリストをクリア。      */
     /*  このリストは後に、計算されたエラー含み、最高から
      最低にソート          */
/*  される。 */

     for(i = 0; i < pixels; i++)  {
               tiles[the_tile].list[i].score=99999999;
               tiles[the_tile].list[i].rank = 0;
     }

     strcpy(imagename, filename);  /*  ターゲット・イメージの
名前を得る     */

imagename[strlen(imagename)-3] = 's'; /* それが適切なファイル名エ
クステンションを有することを確かにする      */
imagename[strlen(imagename)-2] = 'g';
     imagename[strlen(imagename)-1] = 'i';

get_grid_space(rmas, gmas, bmas, x, y); /* ターゲットイメージの
 所望の領域に対するイメージ          */
               /*   データを得て、それを３つのアレイに入れる。
*/

image = head_image;      */  ソースイメージのリンクされたリス
トを開始にリセット       */

while(image->next != NULL) {    /*  各ソースイメージに対して我々
は考慮する          */

     result = 0;
/* これはＲＧＢ ＲＭＳエラーのバリエーションである。処理を高速化
するために最終的二乗根 */
/* は除去されている。我々は、相対的なエラーについてのみ考慮するの
で、これを行うことができる。*/
/* ここで、考慮中のターゲットイメージの部分と視覚的に類似するソー
スイメージを*/
/* 見つける目的が達成される限り、ＨＳＶ ＲＭＳエラー又は他の一致
用のシステムを、使用 */
/* することができる。 */
```

【００１９】　　40　【表３】

```
for(i = 0; i > size; i++) {
      rt   =   (int)((unsigned   char)rmas[i]   -   (unsigned
      char)image->r[i]);
      gt  =  (int)((unsigned char)gmas[i]  -  (unsigned char)
      image->g[i];
      bt   =   (int)((unsigned   char)bmas[i]   -   (unsigned
      char)image->b[i];
      result += (rt*rt+gt*gt+bt*bt);
}
i = 0;

  /* 以下のコードは、最後のソースイメージに対して計算されたエラ
ーを取り、*/
  /* それを、すべてのソースイメージのソートされたリストに挿入す
る。リストは、この挿入のための場所を作るために、*/
  /* 終わりの方に向けてシフトされる。 */

      if (result < tiles[the_tile].list[pixels-1].score) {
                while((result > tiles[the_tile].list[i].score)
&&(i++ < pixels));

      for(ii = pixels-1; ii> i; ii--) {
                tiles[the__tile].list[ii].score  =  tiles[the
                tile].list[ii-1].score;
                tiles[the__tile].list[ii].rank  =  tiles[the
                tile].list[ii-1].rank;
                tiles[the_tile].list[ii].pointer = tiles[the
                tile].list]ii-1].pointer;
}
                tiles[the_tile].list[i].score = result;
                tiles[the_tile].list[i].rank = i;
                tiles[the_tile].list[i].pointer = image;
}
      /*   ここで、次のソースイメージに移り、なくなるま
      で繰り返す */

      image = image->next;

      } /* while (ホワイル) */

/* リストは、次のものから悪いものへとソートされるので、最初のリ
ストのエントリを見ることによって、*/
/* 最高のタイルを見ることができる。*/

low = tiles[the_tile].list[0].score;
tiles[the_tile].score = tiles[the_tile].list[0].score;
tiles[the_tile].rank = tiles[the_tile].list[0].rank;

strcpy(best_path, tiles[the_tile].list[0].pointer->path);

/* このイメージを後に置換しない。なぜなら、モザイクに要求されるも
 のとして、それが指定されていたからである。  */
tiles[the_tile].required = tiles[the_tile].list[0].pointer-
>required;
```

【００２０】

40　【表４】

```
strcpy(tiles[the_tile].path, best_path);
sprintf(imagename, "%s/%s", disp_version, best_path);

/* ここで、ターゲットイメージのこのグリッド位置に対しての、最
も視覚的に類似するものから最も視覚的に類似しないものまでの、*/
/* ソースイメージのソートされたリストを有する。*/

return low;

} /* find_matches () */
```

【００２１】本発明の一実施形態において第２のルーチンが用いられ、ソースされたリストを前のルーチンから取り出し、かつ、各ソース・イメージが１回のみ使用されることを保証するばかりでなく、前記１つの領域に対

して所与のソース・イメージが、もしそれが他の領域において一致（整合）の度合いがより低いならば、選択されないことを調べる。（このプログラムは、オンラインの特許出願の方式に従った１つのイメージ入力範囲内に収まらないので、２つの表（表５、表６）に分割した。）

【００２２】

【表５】

```
        /* プログラム（find_matches ()）の第１のフェーズにおいて、ターゲットイメージの各グリッドスペースに */
        /* 対するソースイメージのソートされたリストを生成した。モザイク内でソースイメージを繰り返したく */
        /* ないので、各グリッドスペースは、その第一選択のソースイメージを有することができない（ソースイ */
        /* メージは、１より多くのグリッド位置に対して最低の一致の度合いを有し得る）。このルーチンの目的は、どのグリッド位置が、*/
        /* ソースイメージを使用するために実際に得るかを決定することである。例えば、それは、別の */
        /* 位置に対する一致の度合いがより高いならば、或るグリッド位置には配置されない。*/

 int optimize ()
 {

        int i, x, deepest = 0, change, a, step, which;

        /* ターゲットイメージにおけるグリッド位置の各々に対する（最終的モザイクにおけるタイルの数）*/
        /* これはN^２アルゴリズムであり、従って、すべてののグリッド位置に対してすべてのイメージを考慮することを確実にするために、２回ループし */
        /* なければならない。*/
        for(a = 0; a < pixels; a++) {
                change = 0;
                /*   ターゲットイメージにおけるグリッド位置の各々に対する（最終的モザイクにおけるタイルの数）*/
 for(x = 0; x < pixels; x++) {
        which = 0;
        do {
                step = 0;
                for(i = 0; i < pixels; i++) {

                                        /* タイルがどこかでまだ所望されるのであれば、それらに与える。*/
                                        /* 他のグリッド位置に対しての最高の選択されたもののすべてを調べることによって、これを行う。*/
                                        /* 別のグリッド位置での第１選択としてリストされた同じソースイメージを見つけると、*/
                                        /* それがその別の位置での一致の度合いがより高いかを見るためにチェックする。*/
                                        /* そうである場合、ソートされたリストを、現在のグリッド位置に対して次に一致の度合いが高いものまで移動し、*/
                                        /* これを、その他のところでは一致の度合いが高くないソースイメージを見つけるま */
                                        /* で行う。これを見つけたとき、それをキープすることができる。変数「ｓｔｅｐ(ステップ)」は０にとどまり、*/
                                        /* ｄｏ－ｗｈｉｌｅ（ドゥー・ホワイル）ループをでる。*/
                                        if ((tiles[i]. rank <= which)  &&  (!strcmp(tiles[x].list[which].pointer->patch, tiles[i].path))) {
```

【００２３】

【表６】

17　　18

```
ば、スコア(score)をチェックする。*/
which)  &&  (tiles[i].score  >  tiles[x].list[which].score))
continue
                                        /*ランク(rank)が同じであれ
                                        if   ((tiles[i].rank   ==
                                        if (i == x) continue;
                                        which++;
                                        step = 1,
                                        i  -  pixels;  /* w h i l e
(ホワイル) へスキップする。*/
                                        }
                }
        } while (step);

        if (which > deepest) deepest = which;

        /*   他のグリッド位置において一致の度合いが高いものではない、最も
視覚的に類似のソース */
        /*   イメージを見つけたので、ターゲットイメージのこのグリッド位置
と関連したイメージの名前 */
        /*   を見る。*/

        if (strcmp(tiles[x].path, tiles [x].list[which].pointer-
>path)) {
        change++;
        strcpy(tiles[x].path,     tiles[x].list[which].pointer-
>path);
        tiles[x].required    =    tiles[x].list[which].pointer-
>required;
        tiles[x].score = tiles[x].list[which].score;
        tiles[x].rank = which;
}

} /* f o r (フォー) */

/*   すべてのグリッド位置を通して調べ、別の位置においてより良く一致する
ものとして */
/*   何れのタイルも置換する必要がなければ、ここでルーチンを出ることがで
きる。 */
if (!change) break;

fprintf(stderr, "\n%d/%d, %d changes (deepest is %d)\n", a,
pixels-1, change, deepest);

/*   最終的モザイクにおけるグリッドスペースがあればその回数だけこのフォー
(f o r)ループでループして戻る必要がある。 */ } /* f o r(フォー) */

} /* optimize() */
```

【００２４】突き合わせプロセスに続いてモザイク・イメージを生成するために、レンダリング・プログラムを用いることができる。レンダリング・プログラムは、選択されたタイルのリストを読み出し、データベースにおける個々の対応するソース・イメージのフル・サイズのバージョンを見つけ出し、見つけ出されたソース・イメージを結合してビット・マップを生成する。モザイク・イメージにおけるタイルは、近い所から見た時に、これらタイルを弁別するために線で分けられる。離れた所からは、モザイクに継ぎ目のないように見せるために、格子線は、人間の目には完全に見えないほど細くなければならない。次に、ビット・マップはモニターに表示されるか又は印刷形態で出力されるように、標準フォーマットでセーブされる。

【００２５】デジタル・モザイク・イメージは、品質、値段およびサイズの制約に従って、異なる方法でプリントすることができる。フィルム記録および写真印刷を用いることもできる。フィルム・レコーダを用いて、イメージを写真フィルムに書き込むことができる。イメージがいったんクローム又はネガティブにのると、通常の印画紙にプリントすることができる。このような選択は、イメージのフィルムへの書き込みの費用は一度だけであるので、適当数の小さなコピーを作る場合に最適である。直接的なデジタル印刷は、最も高い品質を生じるが、各プリントは高価である。デジタル・プリンタは、連続階調あるいは中間調のいずれかを用いる。連続階調プリンタは、イメージにおける各ピクセルに対して正確な色を置く。中間調プリンタは、単色（solid color）の滴（ドロップ）のみを置いて、異なる大きさまたは異なる間隔のドットを用いることにより色の明暗を形成する。従って、プリントは写真らしく見えにくい。プロセス・カラー印刷は、雑誌や書籍においてイメージを再現

するために用いられる技法であり、多数の（例えば、数十万単位の）写真に近いコピーを生成するための良好な方法である。

【００２６】ソース・イメージ選択でのサブ領域を基にする分析の効果が、図５に示されている。第１のモザイク・イメージ１０２、第２のモザイク・イメージ１０４および第３のモザイク・イメージ１０６をそれぞれ生成するため、ターゲット・イメージ１００が用いられた。ターゲット・イメージ１００は４×４タイルを含む。中間の「検知された」イメージは、最小の分析部分（イメージ１０８におけるタイル、およびイメージ１１０および１１２におけるサブ領域）における全てのピクセルの平均を表わす。イメージ１０８および１０２を生じる結果となる第１の分析において、サブ領域は用いられない。イメージ１１０および１０４を生じる結果となる第２の分析において、タイル当たり４×４のサブ領域が用いられる。第２の分析において或る明るい領域と暗い領域が各タイル内で検知できるので、選択プロセスの間にデータベースをサーチする時に、これらの検知された領域が考慮される。

【００２７】イメージ１１２および１０６を生じる結果となる第３の分析においては、１６×１６サブ領域が用いられる。１６×１６サブ領域では、中間イメージ１１２はターゲット・イメージ１００に実質的に近い。更に、イメージ１０６は、選択プロセスの間にこの量のディテール（細部）が考慮される時には、更に適切な整合が選択されるということを示す。例えば、最初の行における女性は、ターゲット・イメージの同じ領域における縦方向の黒いバーと同じ形状である。更に、別のタイルにおけるトカゲは、これが比較されたものの対角線と整合する。このような高度の形状の整合（一致）は、ターゲット・イメージにおける輪郭及び明暗についての情報が各モザイク・タイルの境界を越え得るような、最終的なモザイク・イメージのイメージ形成能力に対して強力な効果を有する。

【００２８】改善されたソース・イメージの選択の提供に加えて、サブ領域を使用し、高周波ディテール（high-frequency detail）の少ない領域に対して低いコントラストのイメージを選択することによって、色が更に一様に分布される。このことはイメージ１０６の下方の８つのタイルで認めることができ、それらはイメージ１０４に対して選択されたものより更に一様である。

【００２９】図６は、モザイク・イメージ解像度における多数のサブ領域数の効果を示している。第１のモザイク・イメージ１１４と第２のモザイク・イメージ１１６は、ターゲット・イメージ１１８から生成された。第１のモザイク・イメージ１１４は、ソース・イメージ選択

プロセスの間に考察された各タイル内の２×２サブ領域により生成された。第２のモザイク・イメージ１１６は、ソース・イメージ選択プロセスの間に考察された各タイル内の１６×１６サブ領域により生成された。第１および第２の両モザイク・イメージを生成するために、ソース・イメージの同じ集まりが用いられた。サブ領域の分析のゆえに、第１および第２のモザイク・イメージで対応するあるタイルを表すために異なるソース・イメージが選択された。更に、第２のモザイク・イメージ１１６は、第１のモザイク・イメージ１１４よりもターゲット・イメージ１１８との類似性が更に強い。従って、更に多くのサブ領域の分析により与えられる向上したソース・イメージの選択が、結果として得るモザイク・イメージにおける向上された解像度を生じる。

【００３０】代替的な実施の形態において、モザイク・イメージの各部に対して意味（semantic）内容が指定される。特定的には、ターゲット・イメージの指定されたタイルと共に使用するためにイメージのサブ領域が指定される。従って、結果的なモザイク・イメージが、予め定めたカテゴリのイメージを持つタイルあるいはタイルの領域を含む。

【００３１】別の代替的な実施の形態では、モザイク・イメージにおいて確保して（確実に）選択および包含するように、イメージを選択することができる。特定的には、選択されたイメージは、たとえ別の（確保されていない）イメージが大きな視覚的類似性を持つと判定されても、ターゲット・イメージに相対的な最も視覚的に類似する場所に置かれる。

【００３２】本発明の望ましい実施形態について述べたが、本発明の概念を包含する他の実施の形態が当業者には明らかであろう。従って、本発明は、開示された実施の形態に限定されると見なされるべきはなく、特許請求の範囲によってのみ限定されるものと見なされるべきである。

【図面の簡単な説明】

【図１】モザイク・イメージ生成システムのブロック図である。

【図２】ソース・イメージのデータベースのブロック図である。

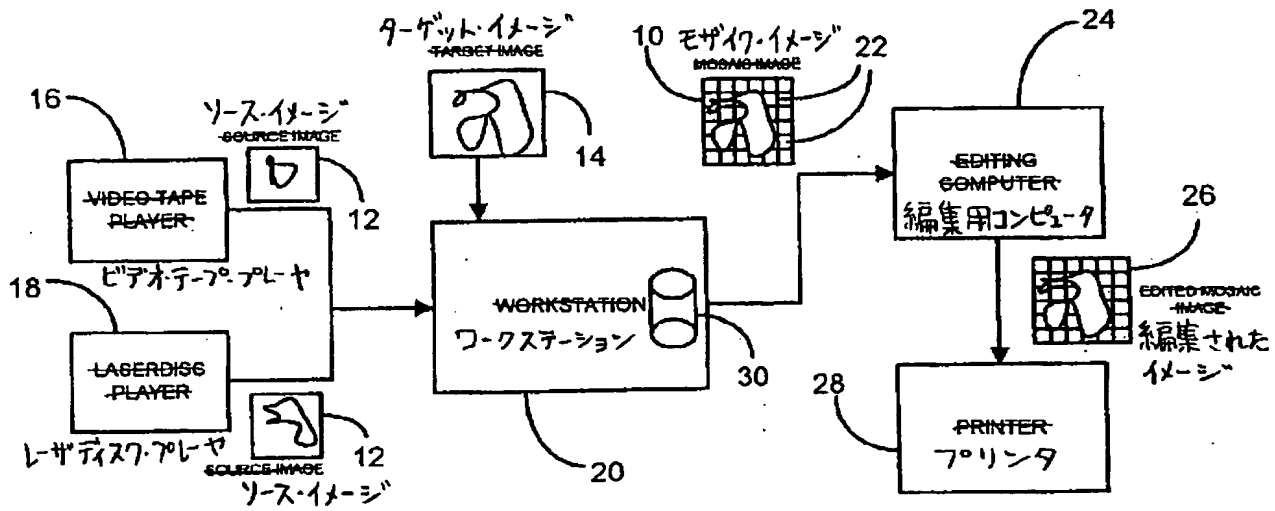
【図３】モザイク・イメージ生成方法を示すフロー図である。

【図４】タイルとサブ領域とを示す図である。

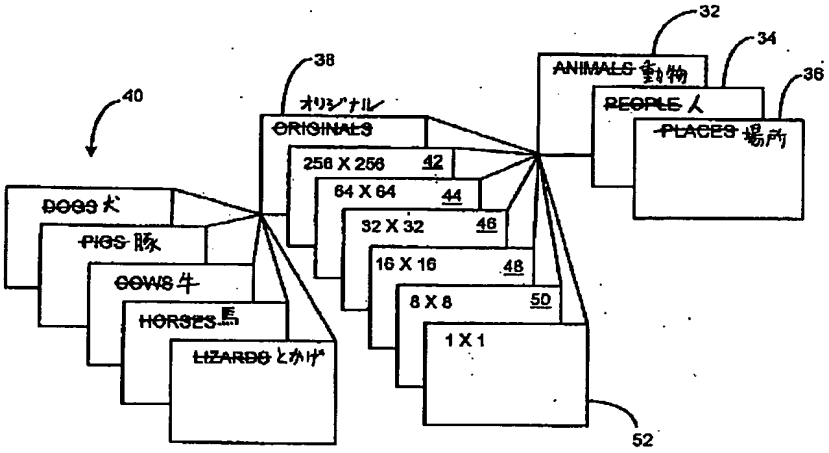
【図５】ソース・イメージの選択についてのサブ領域分析の効果を示す図である。

【図６】最終的なモザイク・イメージの解像度についてのサブ領域分析の効果を示す図である。

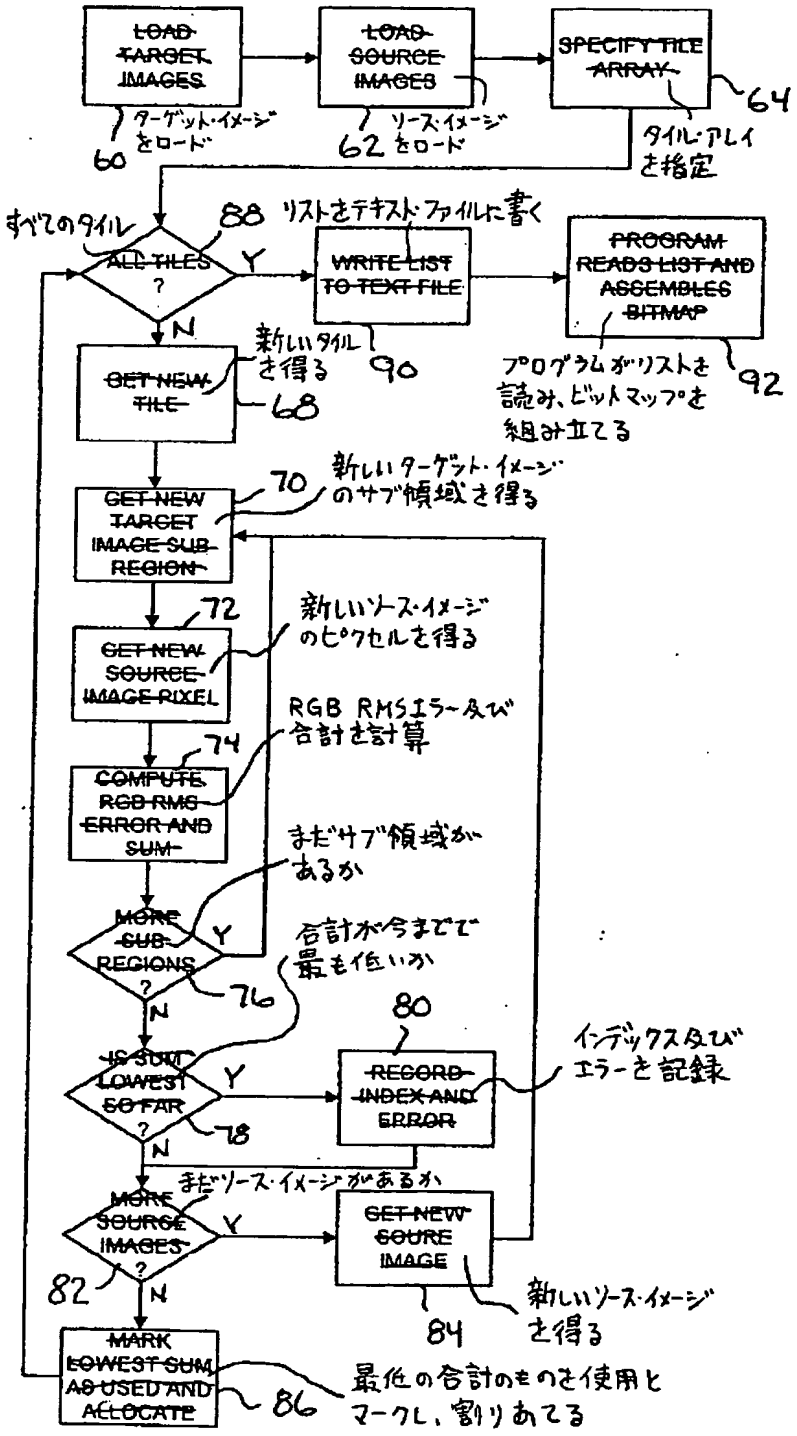
【図１】


ターゲット・イメージ
TARGET IMAGE
10
モザイク・イメージ
MOSAIC IMAGE
22
24
16
ソース・イメージ
SOURCE IMAGE
14
12
EDITING COMPUTER
編集用コンピュータ
26
VIDEO TAPE PLAYER
ビデオテーププレーヤ
18
WORKSTATION
ワークステーション
30
EDITED MOSAIC IMAGE
編集されたイメージ
LASERDISC PLAYER
レーザディスクプレーヤ
28
PRINTER
プリンタ
12
SOURCE IMAGE
ソース・イメージ
20

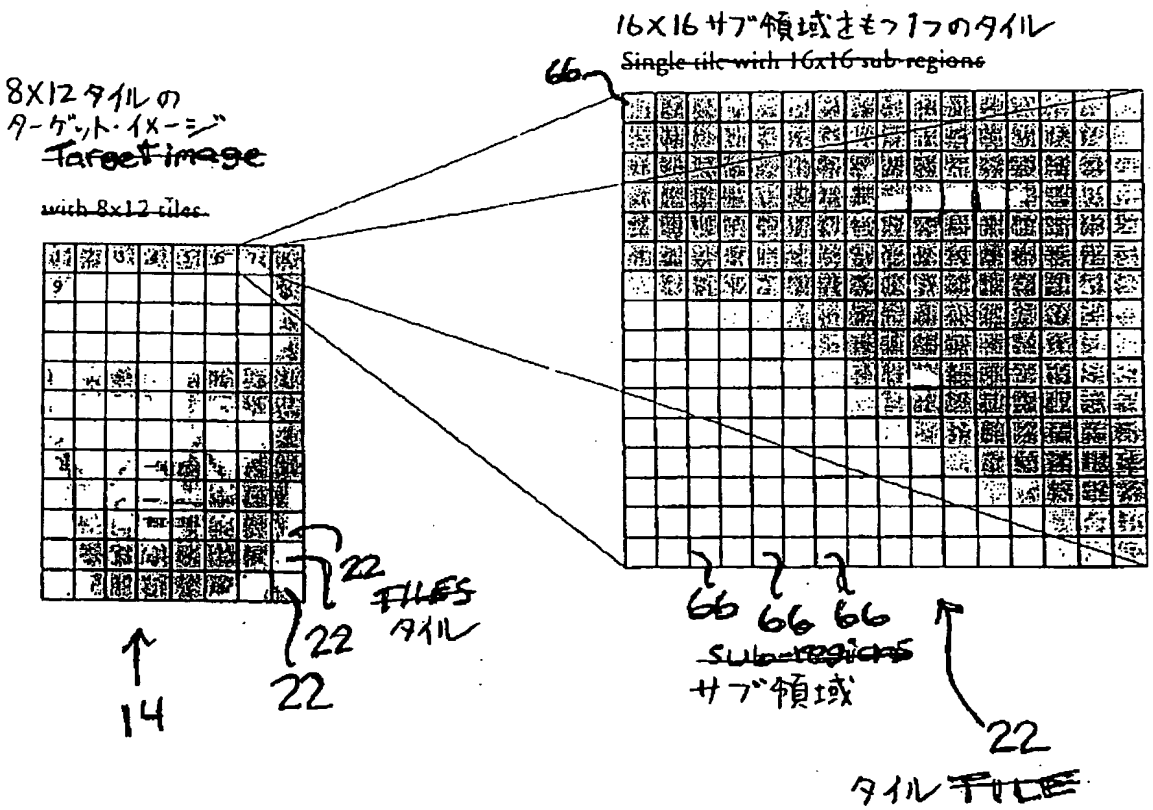
【図２】


32
34
38
ANIMALS 動物
36
40
オリジナル
PEOPLE 人
ORIGINALS
PLACES 場所
256 X 256
42
64 X 64
44
DOGS 犬
32 X 32
46
PIGS 豚
16 X 16
48
COWS 牛
8 X 8
50
HORSES 馬
1 X 1
LIZARDS とかげ
52

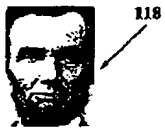
【図３】

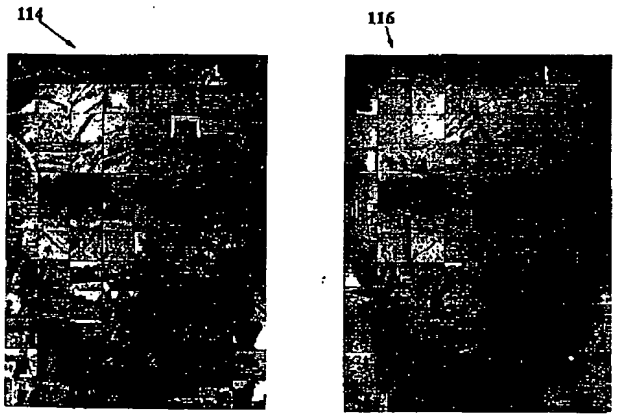

LOAD TARGET IMAGES
ターゲット・イメージをロード
60
LOAD SOURCE IMAGES
ソース・イメージをロード
62
SPECIFY TILE ARRAY
64
タイル・アレイを指定
すべてのタイル
ALL TILES ?
88
Y
N
リストをテキスト・ファイルに書く
WRITE LIST TO TEXT FILE
90
PROGRAM READS LIST AND ASSEMBLES BITMAP
92
プログラムがリストを読み、ビットマップを組み立てる
GET NEW TILE
新しいタイルを得る
68
GET NEW TARGET IMAGE SUB REGION
70
新しいターゲット・イメージのサブ領域を得る
GET NEW SOURCE IMAGE PIXEL
72
新しいソース・イメージのピクセルを得る
COMPUTE RGB RMS ERROR AND SUM
74
RGB RMSエラー及び合計を計算
MORE SUB REGIONS ?
まだサブ領域があるか
76
IS SUM LOWEST SO FAR ?
合計が今までで最も低いか
78
RECORD INDEX AND ERROR
80
インデックス及びエラーを記録
MORE SOURCE IMAGES ?
まだソース・イメージがあるか
82
GET NEW SOURCE IMAGE
84
新しいソース・イメージを得る
MARK LOWEST SUM AS USED AND ALLOCATE
86
最低の合計のものを使用とマークし、割りあてる

【図４】

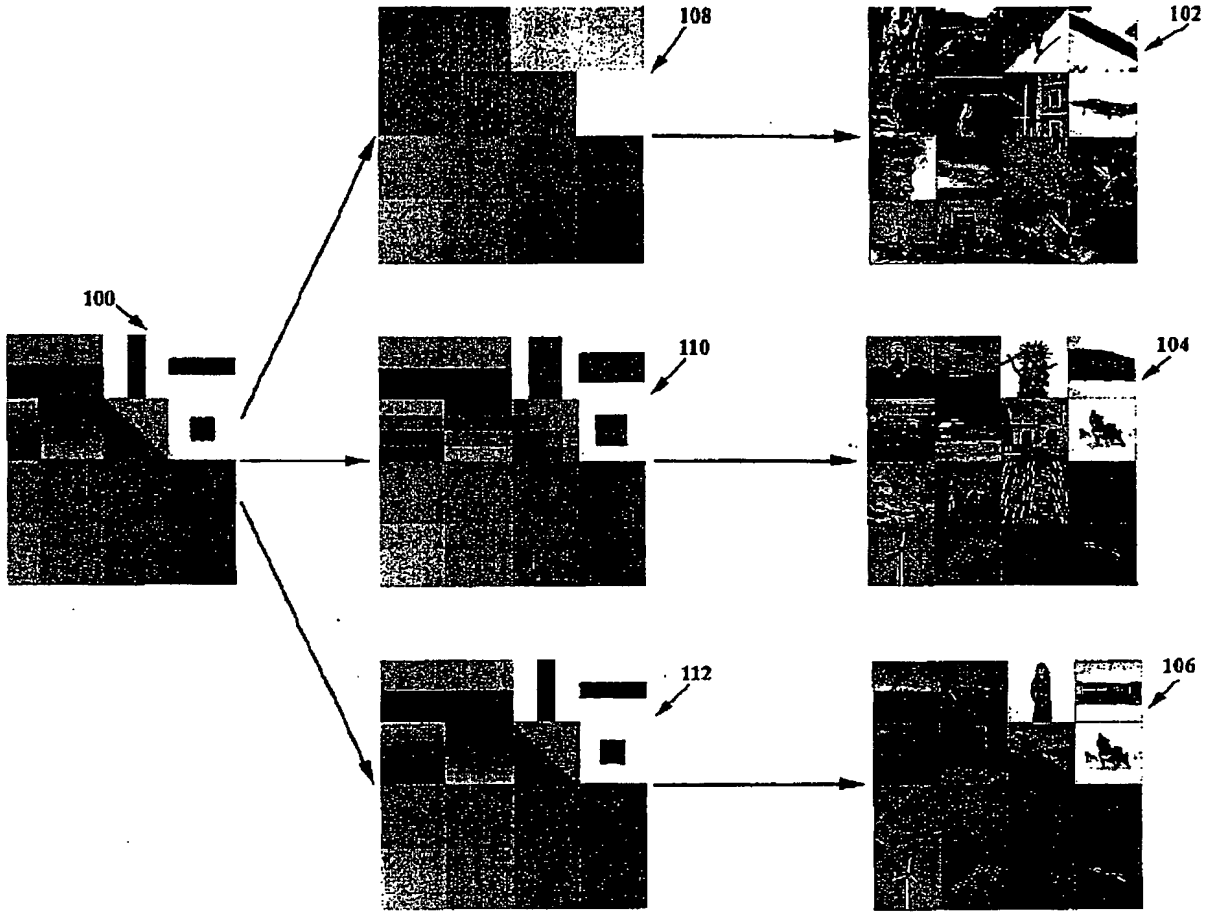

16×16 サブ領域をもつ1つのタイル
Single tile with 16x16 sub-regions
66
8×12 タイルの
ターゲット・イメージ
Target image
with 8x12 tiles
22
TILES
タイル
22
22
14
66
66
66
SUB-REGIONS
サブ領域
22
タイル TILE

【図６】

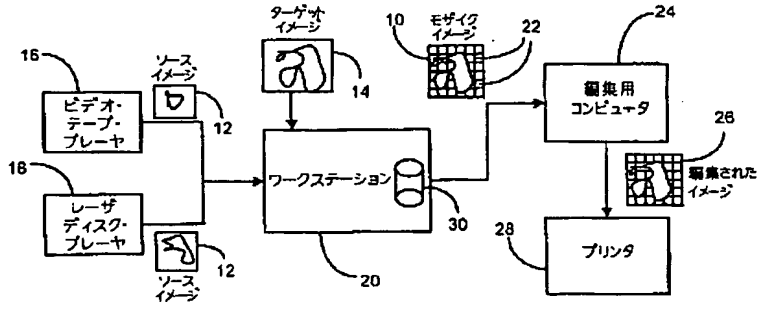

118


114
116

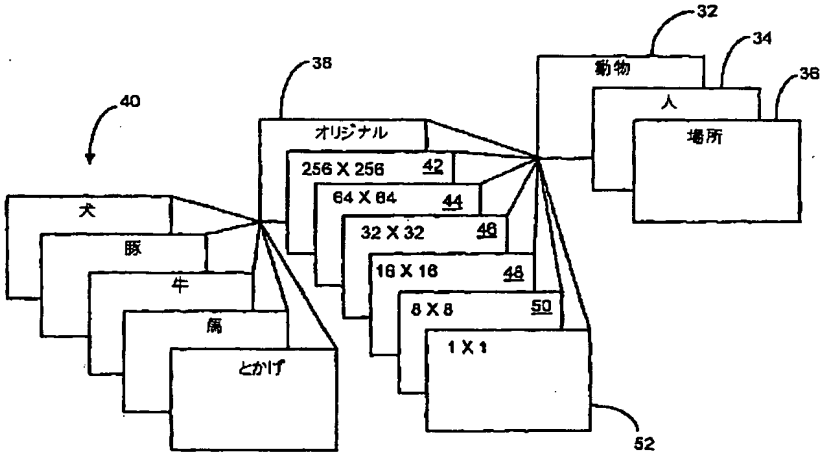
【図５】



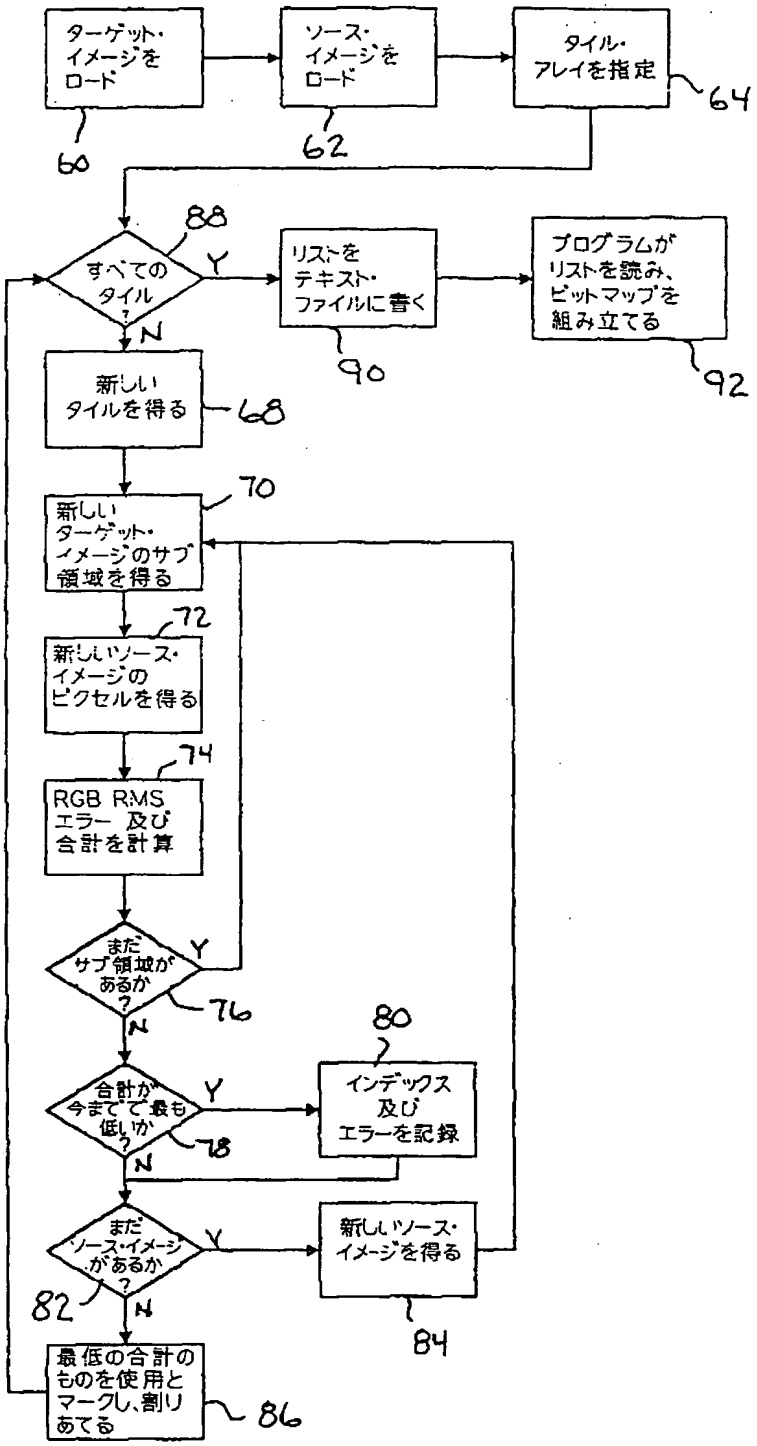
【手続補正書】
【提出日】平成１０年３月１１日
【手続補正１】
【補正対象書類名】図面
【補正対象項目名】図１
【補正方法】変更
【補正内容】
【図１】



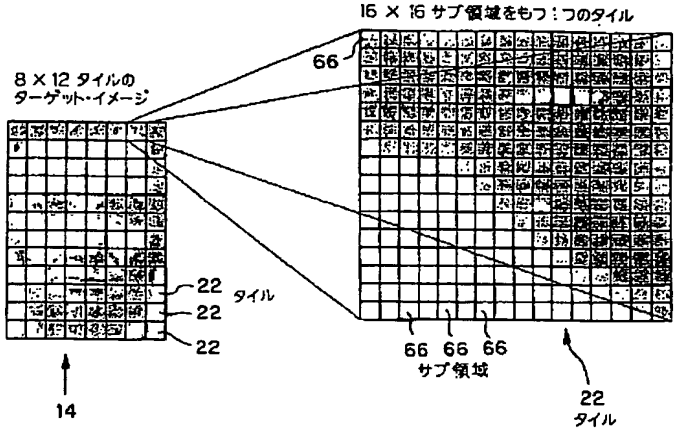
【手続補正２】
【補正対象書類名】図面
【補正対象項目名】図２
【補正方法】変更
【補正内容】
【図２】



【手続補正３】
【補正対象書類名】図面
【補正対象項目名】図３
【補正方法】変更
【補正内容】
【図３】



【手続補正４】
【補正対象書類名】図面
【補正対象項目名】図４
【補正方法】変更
【補正内容】
【図４】



フロントページの続き

(71)出願人　598001434
ランナウェイ・テクノロジー・インコーポレーテッド
Ｒｕｎａｗａｙ　Ｔｅｃｈｎｏｌｏｇｙ，Ｉｎｃ．
アメリカ合衆国マサチューセッツ州02139，ケンブリッジ，メイン・ストリート　875，フォース・フロア
875　Ｍａｉｎ　Ｓｔｒｅｅｔ，４ｔｈ　Ｆｌｏｏｒ，Ｃａｍｂｒｉｄｇｅ，Ｍａｓｓａｃｈｕｓｅｔｔｓ　02139，Ｕｎｉｔｅｄ　Ｓｔａｔｅｓ　ｏｆ　Ａｍｅｒｉｃａ

【外国語明細書】

1. Title of Invention
Digital Composition of a Mosaic Image

2. Claims

1. A method for generating a mosaic image with an appearance that approximates a target image by utilizing a plurality of source images and a computer, comprising the steps of:

loading the target image into the computer;

dividing the target image into a plurality of tile regions, each tile region representing a distinct locus of the target image, and

for each tile region:

comparing source images to the tile region to produce a measurement of visual similarity, said comparing step including analyzing a plurality of individual portions of each source image;

selecting the source image with the highest measurement of visual similarity to represent the tile region; and

positioning the selected source image in the mosaic image at a locus corresponding to the locus of the tile region.

2. The method of claim 1 including the further step of dividing the tile region into distinct sub-regions, each sub-region corresponding to a specific portion of the source image, and comparing each respective sub-region with each respective source image portion to produce the measurement of visual similarity.

3. The method of claim 2 including the further step of employing source images having one pixel per respective sub-region.

4. The method of claim 1 wherein said comparing step includes the further step of computing the average Root-Mean Square error of Red, Green and Blue channels.

5. The method of claim 1 including the further step of removing source images selected in said selecting step from consideration such that no one source image appears more than once in the mosaic image.

6. The method of claim 1 including the further step of capturing source images, and storing the captured source images in a database.

7. The method of claim 6 including the further step of generating modified source images by cropping the source images captured in said capturing step to square.

8. The method of claim 7 including the further step of, in the case of a captured source image in landscape format, cropping the captured image from center.

9. The method of claim 8 including the further step of, in the case of a captured source image in portrait format, cropping the captured image from above center.

10. The method of claim 7 including the further step of categorizing the captured source images within the database.

11. The method of claim 7 including the further step of storing the captured source images at different levels of resolution.

12. The method of claim 1 including the further step of deselecting the source image with the highest measurement of visual similarity if it is determined that the source image has a higher measurement of visual similarity to another tile region.

13. The method of claim 1 including the further step of specifying at least one source image for assured inclusion in the mosaic image, the assured source image being

positioned in the mosaic image at a locus corresponding to the locus of the tile region having the highest measure of visual similarity therewith.

14. The method of claim 1 including the further step of specifying a sub-category of source images for exclusive matching with a predetermined portion of the target image.

15. An apparatus for generating a mosaic image with an appearance that approximates a target image by utilizing a plurality of source images, comprising:

A computer workstation that executes mosaic generation software being operative to divide the target image into a plurality of tile regions, each tile region representing a distinct locus of the target image,

said mosaic generation software being further operative to operate upon each tile region to:

compare a plurality of source image portions to the tile region to produce a measurement of visual similarity;

select the source image with the highest measurement of visual similarity to represent the tile region; and

position the selected source image in the mosaic image at a locus corresponding to the locus of the tile region.

16. The apparatus of claim 15 wherein the mosaic generation software is further operative to divide the tile region into distinct sub-regions, each sub-region corresponding to a specific portion of the source image, each respective sub-region being compared with each respective source image portion to produce the measurement of visual similarity.

17. The apparatus of claim 16 wherein the source image employed for comparison with the tile region has one pixel per respective sub-region.

18. The apparatus of claim 15 wherein the mosaic generation software is further operative to compute the average Root-Mean Square error of Red, Green and Blue channels.

19. The apparatus of claim 15 wherein the mosaic generation software is further operative to remove selected source images selected from consideration such that no one source image appears more than once in the mosaic image.

20. The apparatus of claim 15 further including video equipment selected from the group consisting of a video tape player and a videodisc player, said video equipment being operative to capture source images for storage in a database in the computer workstation.

21. The apparatus of claim 20 wherein modified source images are generated by cropping and resizing the captured source images to a consistent size.

22. The apparatus of claim 21 wherein, in the case of a captured source image in landscape format, the captured image is cropped from center.

23. The apparatus of claim 22 wherein, in the case of a captured source image in portrait format, the captured image is cropped from above center.

24. The apparatus of claim 21 wherein the captured source images are categorized within the database.

25. The apparatus of claim 21 wherein the captured source images are stored at different levels of resolution.

26. The apparatus of claim 20 further including an editing computer with software for editing the mosaic image.

27. The apparatus of claim 26 further including a printer for printing the edited mosaic image.

28. The apparatus of claim 15 wherein the source image with the highest measurement of visual similarity is deselected if it is determined that the source image has a higher measurement of visual similarity to another tile region.

29. The apparatus of claim 15 wherein at least one source image is assured inclusion in the mosaic image, the assured source image being positioned in the mosaic image at a locus corresponding to the locus of the tile region having the highest measure of visual similarity therewith.

30. The apparatus of claim 15 wherein a sub-category of source images is specified for exclusive matching with a predetermined portion of the target image.

3. Detailed Description of Invention

Field of invention:

The present invention is generally related to computerized manipulation of images, and more particularly to generation of an image from a plurality of sub-images.

Prior arts:

Analysis and manipulation of images using computers is well known. For example, computers have been used to analyze images of coins travelling along a conveyor belt to distinguish different types of coins and compute the total value of the coins. Similarly, computers have been used to analyze images of integrated circuits and printed circuit boards in order to detect defects during manufacturing. Manipulation of photographic still images and full motion video images to produce special effects is also well known. However, these known techniques do not produce artistically pleasing mosaic images.

Means to solve problems:

In accordance with the present invention, a mosaic image that approximates a target image is produced from a database of source images by analyzing tile portions of the target image, comparing each respective analyzed tile portion of the target image with the source images from the database to provide a best-fit match in accordance with predetermined criteria, and generating a mosaic image comprising the respective best-fit match source images positioned at respective tile portions of the mosaic image which correspond to the respective analyzed tile portions of the target image. In one embodiment the criteria for the best-fit match includes computing a version of Red, Green and Blue ("RGB") Root-Mean Square ("RMS") error. Other matching systems could be employed as long as the goal of finding the source image that is most visually similar to the region of the target image under consideration is met.

Increased resolution is realized in the mosaic image through sub-region analysis. In particular, each tile portion in the target image is divided into sub-regions which are independently compared with corresponding sub-regions of each source image using, in this example, RGB RMS error analysis. The computed RGB RMS error for each sub-region is summed to provide a sum RGB RMS error for the entire source image. The unallocated image having the lowest sum RGB RMS error is then allocated for use in the corresponding tile portion in the mosaic image. The use of sub-regions even benefits regions without detail and results in more uniform distribution of color by selecting lower contrast images for these areas of little high-frequency detail. Another embodiment employs a second pass to prevent a source image from being placed in a given location in the mosaic if it would have a lower error in another location.

Embodiment:

Fig. 1 illustrates apparatus for generating a mosaic image 10 from captured source images 12 to approximate a target image 14. In the disclosed embodiment a VHS video tape player 16 is employed to facilitate capture of source images from video tapes. The video tape player may be employed to single-step through a video tape to capture still images for use as source images. Alternatively, source images can be captured in real-time during playback of a video tape. A computer controllable laserdisc player 18 also can be employed to facilitate capture of source images. Laserdiscs are preferable to video tapes when the desired subject matter is available from both sources because of the higher quality and easy random access to still images available from laserdisc. In the disclosed embodiment a computer workstation 20 with a video input is employed to capture the source images 12 from the video tape player 16 and laserdisc player 18. The computer workstation also accepts the target image 14 as input, and is employed to

generate the mosaic image 10 from the target image and source images by executing mosaic software. The mosaic image 10 generated by the mosaic software comprises an array of tiles 22, where each tile 22 is a source image 12, and the overall appearance of the mosaic image 10 approximates the appearance of the target image 14. An editing computer 24 such as a Macintosh (TM), PC or UNIX (TM) based system equipped with image editing software such as Adobe Photoshop (TM) can be employed for editing the mosaic image 22, to produce an edited mosaic image 26. A printer output device 28 may be employed to print the edited mosaic image 26.

Captured source images 12 can be analyzed and stored in a database 30 that is maintained in the workstation 20. An add_images_to_database program is employed to analyze raw captured source images 12 and create new source images therefrom. More particularly, the add_images_to_database program accepts a list of filesystem directories, an image size, and an output path as input, and operates in response to open each designated directory and search for source images from which to crop and resize to the specified dimensions. The square is subsequently moved to the location specified by the output path. In one embodiment, if the source image is in landscape format, a square image is cropped from the center of the source image. If the source image is in portrait format, a square is cropped from between the center and the top of the source image. As a consequence, the square image is more likely to include the emphasized feature of the source image, such as a person's face, without clipping the edges thereof. The images are then stored in the database 30. The database 30 is a file system which holds the formatted images in directories that are categorized by subject matter and size.

Fig. 2 illustrates organization of source images 12 within the database 30 (Fig. 1). Source images 12 are categorized and placed under root nodes such as an animals root node 32, a people root node 34 or a places root node 36. To generate a mosaic image from source images of animals, the

animals root node 32 is selected for the mosaic software. Directly under the animals root node are subdirectories containing identical image files at different levels of resolution. An originals subdirectory 38 contains uncropped versions of each source image file at full size 40. The originals subdirectory 38 is maintained because source images may be recropped during mosaic creation if the results from the add_images_to_database program are unacceptable. Directories labeled 256 x 256 (pixels) 42 and 64 x 64 (pixels) 44 contain large versions of the formatted source images which are used primarily for outputting a final bitmap. In this example, a 32 x 32 (pixels) 46 directory contains source images which are used for viewing the mosaic image on the screen during the construction process. The 16 x 16 (pixels) 48, 8 x 8 (pixels) 50, and 1 x 1 (pixels) 52 subdirectories contain source images which are preloaded when the mosaic software is initialized. The source images in the 16 x 16, 8 x 8, and 1 x 1 subdirectories are employed for matching source images to target image during mosaic image generation. Directories of source images at other levels of resolution may also be maintained.

Fig. 3 illustrates a method for generating the mosaic image. Referring now to Figs. 2, 3 and 4, the target image is selected and loaded as indicated in step 60. A root node of source images in the database is then selected and loaded as indicated in step 62. More particularly, a database path is specified and a mosaic program is executed. The mosaic program reads source images from the section of the database indicated by the specified database path, analyzes the target image and selects a source image for use in each tile of the mosaic image. More particularly, source images having resolution corresponding to the selected number of sub-regions ("sub-region resolution") for the mosaic image are loaded into a linked list of structures:

```
struct an_image {
    char *path;                    /pathname of file in database/
```

```
    char used;                    /whether image has been
used/
    unsigned short *r;            /RGB image data for RMS
matching/
    unsigned short *g;
    unsigned short *b;
    struct an_image *next;        /pointer to next
structure/
    struct an_image *previous; /pointer to prev structure/
} an_image;
```

For example, if each tile in the mosaic image is to contain 8 X-axis sub-regions by 8 Y-axis sub-regions, then 8 X 8 (pixels) images are loaded from the database. The size of the target image in pixels along each axis is equal to the number of output tiles multiplied by the number of desired sub-regions to be considered during the matching process, i.e., one pixel per sub-region along each respective axis. The respective numbers of tiles which will be employed for the X and Y axes of both the mosaic image and target image is then specified as indicated in step 64.

The mosaic program executes a matching process once the source and target images have been loaded. When the matching process begins, the target image is divided into "x" by "y" tiles 22, where (x, y) is:

```
(target_image_width / width_subsamples, target_image_height
/ height_subsamples)
```

A new tile is loaded as indicated in step 68. A new sub-region 66 is then loaded as indicated in step 70. Loading begins with the upper left sub-region 66 of the tile 22, and moves from left to right through each row, and from top to bottom by row. The source image pixel that corresponds to the loaded sub-region is then loaded as indicated in step 72.

The matching process analyzes tiles 22 individually on a serial basis. For each tile 22 in the disclosed

embodiment, a variation of the average Root-Mean Square ("RMS") error of the Red, Green, and Blue ("RGB") channels of each sub-region 66 is compared to each respective corresponding source image pixel, for each source image in the database that is of proper resolution and is not designated as "used." A RMS error between the loaded pixel and loaded sub-region is computed for RGB channels and kept as a running sum for the tile as indicated in step 74. If unanalyzed sub-regions exist in the tile as indicated in step 76, flow returns to step 70. If all sub-regions have been analyzed, as determined in step 76, then the running sum RGB RMS error is compared to the lowest such error yet computed for a source image and the tile as indicated in step 78. If the error sum is lower than any previously recorded error sum for the tile, the error sum value and an index to the source image are recorded as indicated in step 80.

When all of the source images have been analyzed for similarity to the tile, the source image with the least computed RGB RMS error is assigned to a tile in the mosaic image corresponding to the tile in the target image, i.e., in the same location in the image. More particularly, if other source images in the database have not been compared with the tile as determined in step 82, a new source image is loaded as indicated in step 84 and flow returns to step 70. If all source images have been compared with the tile as determined in step 82, the source image with the lowest sum error is allocated to the tile and marked as "used" as indicated in step 86. The assigned source image is marked as "used" so that source images do not appear more than once in the mosaic image.

The matching process is repeated for each and every tile in the target image. Upon completion, a list of source images is written to a text file which is used by a final rendering program to construct a bitmap from the full resolution versions of the source images. More particularly, if all tiles have been examined as determined in step 88, a list of the lowest sum error source images for each tile is

written to a text file as indicated in step 90, and the mosaic program reads the list and assembles a bitmap as indicated in step 92. If unexamined tiles still exist as determined in step 88, flow returns to step 68.

A variation of the matching process, including computation of RMS error, is implemented as follows:

```
/* The goal of this routine is to find which source
photographs are the most */
/* visually similar to a given region (grid-space) of the
target image. */

int find_matches(int x, int y)
{
     register i, rt, gt, bt;
     int low, result, ii, the tile;
     char imagename[256], best_path[256];
     unsigned short rmas[XMAX*YMAX], gmas[XMAX*YMAX], bmas
[XMAX*YMAX];

     the_tile = x+(y*sizex);

     /* For this given grid-location of the target image,
clear the list of errors. */
     /* This list will later contain the computed errors and
will be sorted from best */
/* to worst */

     for(i = 0; i < pixels; i++)  {
               tiles[the_tile].list[i].score=99999999;
               tiles[the_tile].list[i].rank = 0;
     }

     strcpy(imagename, filename); /* Get the name of the
target image */

imagename[strlen(imagename)-3] = 's'; /* Make sure that it
has the proper filename extension */
imagename[strlen(imagename)-2] = 'g';
     imagename[strlen(imagename)-1] = 'i';

get_grid_space(rmas, gmas, bmas, x, y); /* Get the image data
for the desired region of the */
               /* target image and put it into three arrays.
*/

image = head_image;     */ Reset the linked-list of source
images to the beginning */

while(image->next != NULL) {    /* For every source image we
are considering */

     result = 0;
```

```
/* This is a variation of RGB RMS error. The final square-
root has been eliminated to */
/* speed up the process.  We can do this because we only care
about relative error. */
/* HSV RMS error or other matching systems could be used
here, as long as the goal of */
/* finding source images that are visually similar to the
portion of the target image */
/* under consideration is met. */

for(i = 0; i > size; i++) {
     rt   =   (int)((unsigned   char)rmas[i]   -   (unsigned
     char)image->r[i]);
     gt  =  (int)((unsigned char)gmas[i]  -  (unsigned char)
     image->g[i];
     bt   =   (int)((unsigned   char)bmas[i]   -   (unsigned
     char)image->b[i];
     result += (rt*rt+gt*gt+bt*bt);
}
i = 0;

     /* The following code takes the error computed for the
last source image and inserts */
     /* it into a sorted list of all of the source images.
The list is shifted towards the */
     /* end to make room for this insertion */

     if (result < tiles[the_tile].list[pixels-1].score) {
               while((result > tiles[the_tile].list[i].score)
&&(i++ < pixels));

     for(ii = pixels-1; ii> i; ii--) {
               tiles[the_tile].list[ii].score  =  tiles[the
               tile].list[ii-1].score;
               tiles[the_tile].list[ii].rank  =  tiles[the
               tile].list[ii-1].rank;
               tiles[the_tile].list[ii].pointer = tiles[the
               tile].list]ii-1].pointer;
}
               tiles[the_tile].list[i].score = result;
               tiles[the_tile].list[i].rank = i;
               tiles[the_tile].list[i].pointer = image;
}
     /* Now let's move to the next source image and repeat
     until we run out */

     image = image->next;

     } /* while */

/* Since the list is sorted from next to worse, we can see
the best tile by looking at */
/* the first list entry. */

low = tiles[the_tile].list[0].score;
tiles[the_tile].score = tiles[the_tile].list[0].score;
```

```
tiles[the_tile].rank = tiles[the_tile].list[0].rank;

strcpy(best_path, tiles[the_tile].list[0].pointer->path);

/* Do not let this image get replaced later because it was
specified as required for the mosaic.*/
tiles[the_tile].required = tiles[the_tile].list[0].pointer-
>required;

strcpy(tiles[the_tile].path, best_path);
sprintf(imagename, "%s/%s", disp_version, best_path);

/* We now have a sorted list of source images from most-
visually-similar to least-visually-similar */
/* for this grid location of the target image.*/

return low;

} /* find_matches () */
```

A second routine is used in one embodiment of the invention to take the sorted list from the previous routine and not only ensure that each source image is only used once but also to see that a given source image will not be selected for one region if it is an even lower match in another.

```
     /* In the first phase of the program (find_matches ()),
e created a sorted list of source images */
     /* for each grid-space of the target image.  Since we
do not want to repeat source images within */
     /* the mosaic, each grid-space cannot have its first
choice source image (a source image may have */
     /* the lowest match for more than one grid location).
The purpose of this routine is to decide which */
     /* of the grid locations actually gets to use the source
image.  For example, it will not be placed */
     /* in one grid location if it an even better match to
another */

int optimize ()
{

     int i, x, deepest = 0, change, a, step, which;

     /* For each of the grid-locations in the target image
(number of tiles in the final mosaic) */
     /* This an N^2 algorithm, so we must loop twice to
ensure that we consider all images for */
     /* all grid-locations. */
     for(a = 0; a < pixels; a++) {
               change = 0;
```

```
                /* For each of the grid-locations in the
target image (number of tiles in the final mosaic) */
for(x = 0; x < pixels; x++) {
     which = 0;
     do {
                step = 0;
                for(i = 0; i < pixels; i++) {

                                        /* If tile is wanted more
somewhere else, give it to them. */
                                        /* We do this by going
through all the top choices for the other grid locations. */
                                        /* If we see the same
source image listed as the first choice at another grid */
                                        /* location, we check to
see if it is a better match at the other location. */
                                        /* If it is, we move
through our sorted list to the next best match for our
current */
                                        /* grid-location and do
this until we find a source image that is not a better match
*/
                                        /* anywhere else.  When
we find this, we can keep it.  The variable "step" stays as
0 and */
                                        /* we exit the do-while
loop */
                                        if ((tiles[i]. rank <=
which)  &&  (!strcmp(tiles[x].list[which].pointer->patch,
tiles[i].path))) {
                                        /* If rank is same, check
scores. */
                                        if  ((tiles[i].rank  ==
which)  &&  (tiles[i].score  >  tiles[x].list[which].score))
continue
                                        if (i == x) continue;
                                        which++;
                                        step = 1,
                                        i - pixels; /* Skip to
while. */
                                        }
                }
     } while (step);

     if (which > deepest) deepest = which;

     /* Now that we found the most visually-similar source
image that is *not* a better match in another */
     /* grid location, we se the name of the image as
associated with this grid-location of the target */
     /* image. */

     if (strcmp(tiles[x].path, tiles [x].list[which].pointer-
>path)) {
     change++;
     strcpy(tiles[x].path,    tiles[x].list[which].pointer-
```

```
>path);
     tiles[x].required    =    tiles[x].list[which].pointer-
>required;
     tiles[x].score = tiles[x].list[which].score;
     tiles[x].rank = which;
}

} /* for */

/* If we go through all of the grid-locations and we do not
need to replace any */
/* tiles as being a better match in another location, we can
exit the routine now. */
if (!change) break;

fprintf(stderr, "\n%d/%d, %d changes (deepest is %d)\n", a,
pixels-1, change, deepest);

/* We need to loop back with this for loop as many times as
there are grid-space in the final mosaic. */ } /* for */

} /* optimize() */
```

A rendering program can be employed to produce the mosaic image following the matching process. The rendering program reads the list of the selected tiles, locates the full sized version of each respective corresponding source image in the database, and binds the located source images together to create a bitmap. The tiles in the mosaic image may be separated by a line to discretize them when viewed from close proximity. From a distance, the gridlines should be thin enough to disappear completely to the human eye, so as not to interfere with the seamlessness of the mosaic. The bitmap is then saved in a standard format to be displayed on a monitor or output in printed form.

The digital mosaic image can be printed in different ways, depending on quality, price and size constraints. Film recording and photographic printing may be employed. An image can be written to photographic film using a film recorder. Once the image is on chrome or negative, it can be printed on normal photographic paper. This option is best for a moderate number of small copies as writing the image onto the film is a one time cost. Direct digital printing potentially produces the highest quality, but each print is expensive. Digital printers employ either continuous-toning

or half-toning. Continuous-tone printers deposit an exact color for each pixel in the image. Half toning printers deposit only drops of solid color, forming shades of color by using dots of different sizes or different spacing. Hence, the print will look less photographic. Process color printing is the technique used to reproduce images in magazines and books, and is a good method for producing many (e.g., hundreds of thousands) near-photographic copies.

The effects of sub-region based analysis on source image selection are illustrated in Fig. 5. A target image 100 was employed to produce first, second and third mosaic images 102, 104, 106, respectively. The target image 100 includes 4 X 4 tiles. An intermediate "sensed" image representing the average of all pixels in the smallest analyzed portion (tiles in image 108, and sub-regions in images 110 and 112). In the first analysis, resulting in images 108 and 102, sub-regions are not employed. In the second analysis, resulting in images 110 and 104, 4 X 4 sub-regions per tile are employed. Because some light and dark regions can be sensed within each tile in the second analysis, those sensed regions are taken into consideration when searching the database during the selection process.

In the third analysis, resulting in images 112 and 106, 16 X 16 sub-regions are employed. With 16 X 16 sub-regions, the intermediate image 112 is substantially closer to the target image 100. Further, image 106 shows that when this amount of detail is considered during the selection process, more appropriate matches are selected. For example, the woman in the first row is the same shape as the vertical black bar in the same region of the target image. Further, the lizard in another tile matches the diagonal that it was compared to. This high-degree of shape matching has a powerful effect on the image-forming ability of the final mosaic image as information about the contours and shading in a target image may transcend the boundaries of each mosaic tile.

In addition to providing improved source image

selection, the use of sub-regions results in more uniform distribution of color by selecting lower contrast images for regions of little high-frequency detail. This can be seen in the lower eight tiles of image 106 which are more uniform than those selected for image 104.

Fig. 6 illustrates the effects of number of sub-regions on mosaic image resolution. First and second mosaic images 144, 116 were generated from a target image 118. The first mosaic image 114 was generated with 2 x 2 sub-regions within each tile considered during the source image selection process. The second mosaic image 116 was generated with 16 x 16 sub-regions within each tile considered during the source image selection process. The same collection of source images was employed to produce both the first and second mosaic images. Because of the sub-region analysis, different source images were selected to represent some corresponding tiles in the first and second mosaic images. Further, the second mosaic image 116 bears a stronger resemblance to the target image 118 than the first mosaic image 114. Hence, improved source image selection provided through analysis of more sub-regions generates improved resolution in the resultant mosaic image.

In an alternative embodiment, semantic content is specified for portions of the mosaic image. More particularly, image sub-categories are specified for use with specified tiles of the target image. Hence, the resultant mosaic image includes tiles or regions of tiles with predetermined categories of images.

In another alternative embodiment images can be selected for assured selection and inclusion in the mosaic image. More particularly, the selected images are placed in the location of greatest visual similarity relative to the target image even if another (unassured) image is determined to have greater visual similarity.

Having described the preferred embodiments of the invention, other embodiments which incorporate concepts of the invention will now become apparent to one of skill in the

art. Therefore, the invention should not be viewed as limited to the disclosed embodiments but rather should be viewed as limited only by the spirit and scope of the appended claims.

4. Brief description of Drawings

Fig. 1 is a block diagram of a mosaic image generating system;

Fig. 2 is a block diagram of a database of source images;

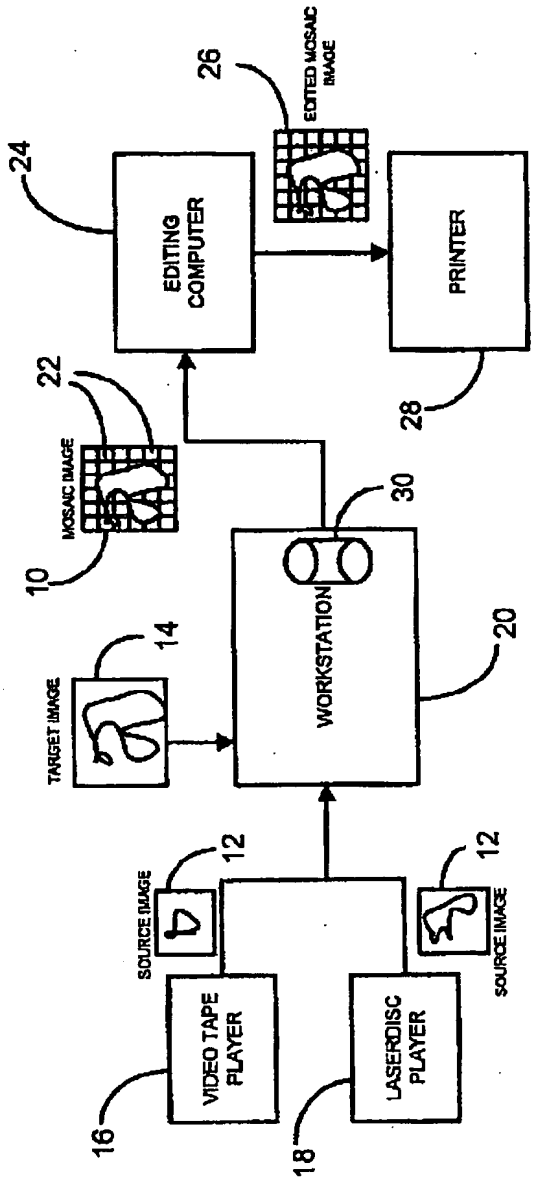
Fig. 3 is a flow diagram that illustrates a method of mosaic image generation;

Fig. 4 is a diagram that illustrates tiles and sub-regions;

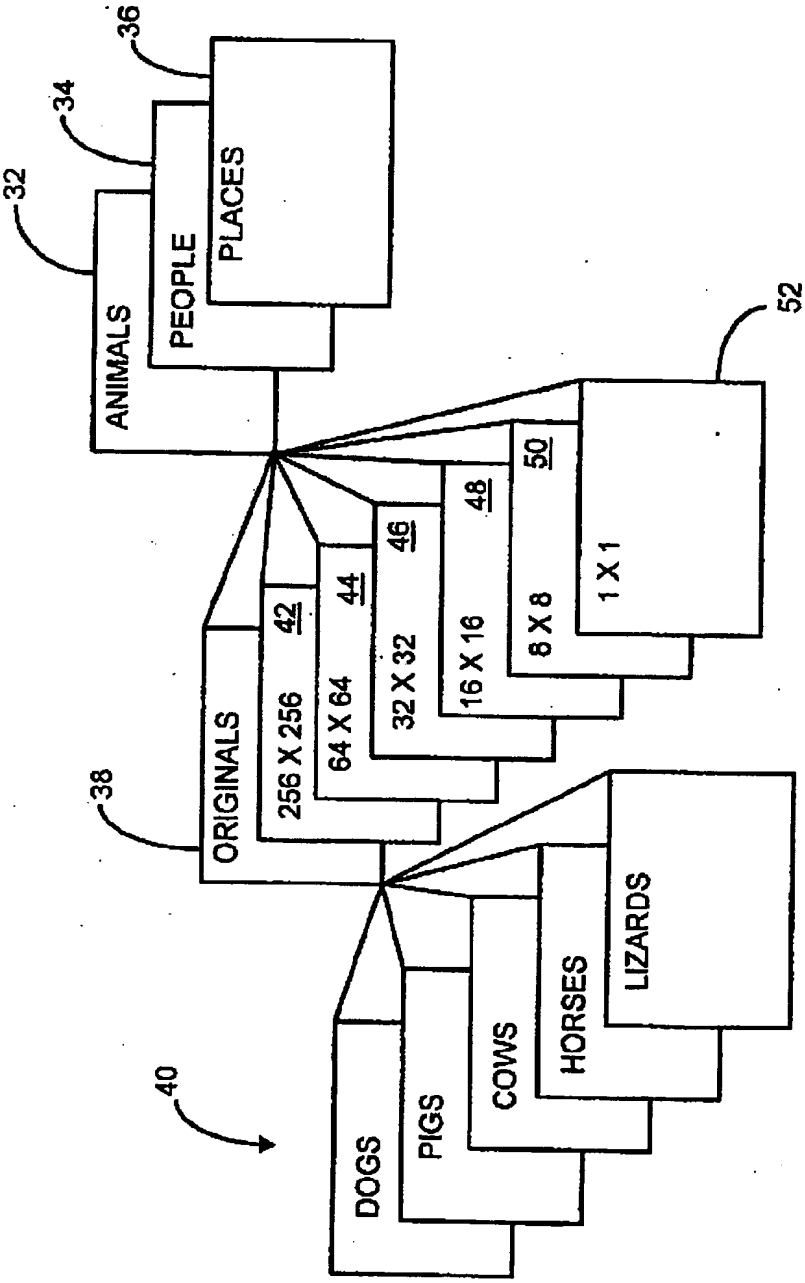
Fig. 5 illustrates the effect of sub-region analysis on source image selection; and

Fig. 6 illustrates the effect of sub-region analysis on final mosaic image resolution.

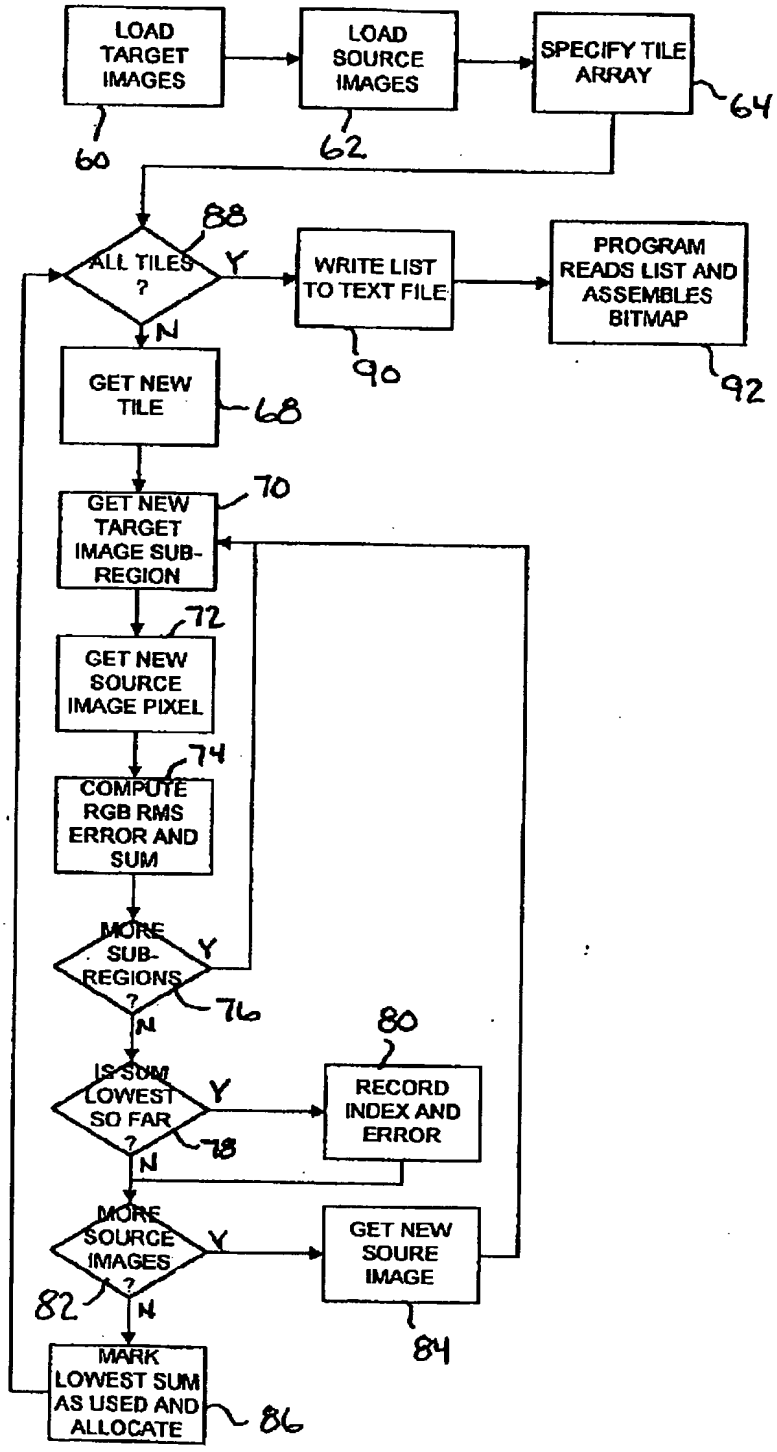
【図１】


EDITED MOSAIC IMAGE
26
24
EDITING COMPUTER
PRINTER
22
28
MOSAIC IMAGE
10
30
WORKSTATION
20
TARGET IMAGE
14
SOURCE IMAGE
12
SOURCE IMAGE
12
VIDEO TAPE PLAYER
LASERDISC PLAYER
16
18

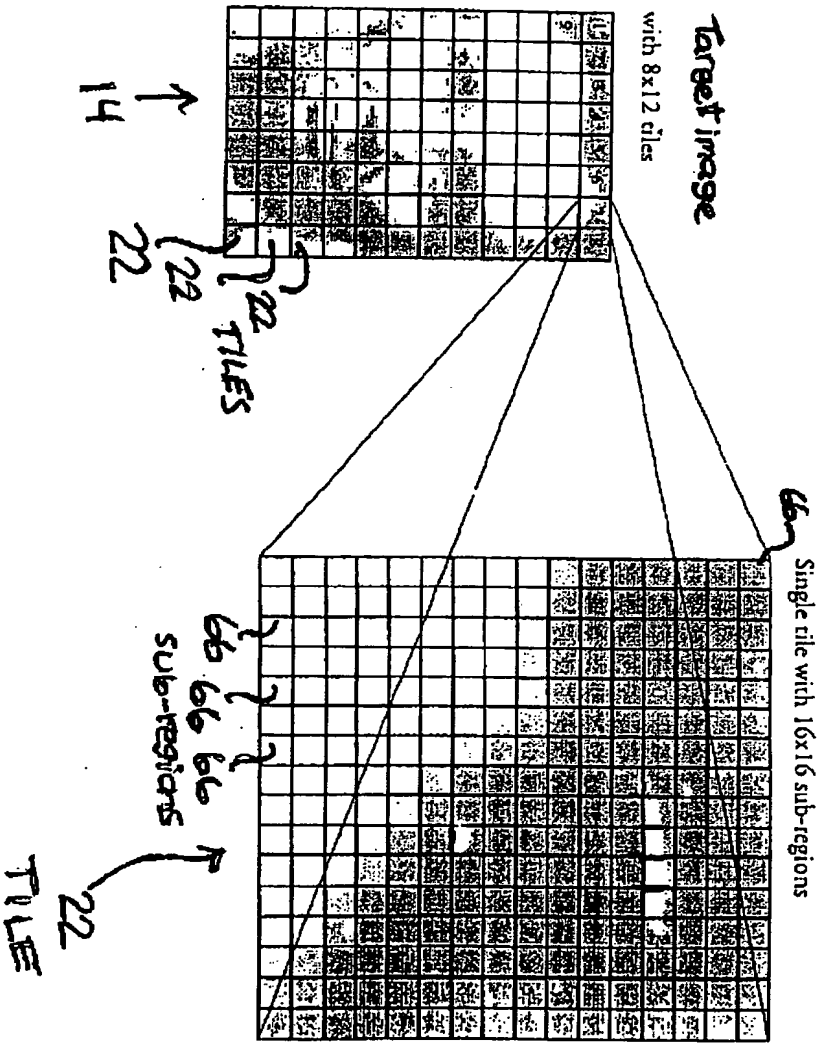
【図２】


36
34
32
PLACES
PEOPLE
ANIMALS
52
50
48
46
44
42
1 X 1
8 X 8
16 X 16
32 X 32
64 X 64
256 X 256
ORIGINALS
38
LIZARDS
HORSES
COWS
PIGS
DOGS
40

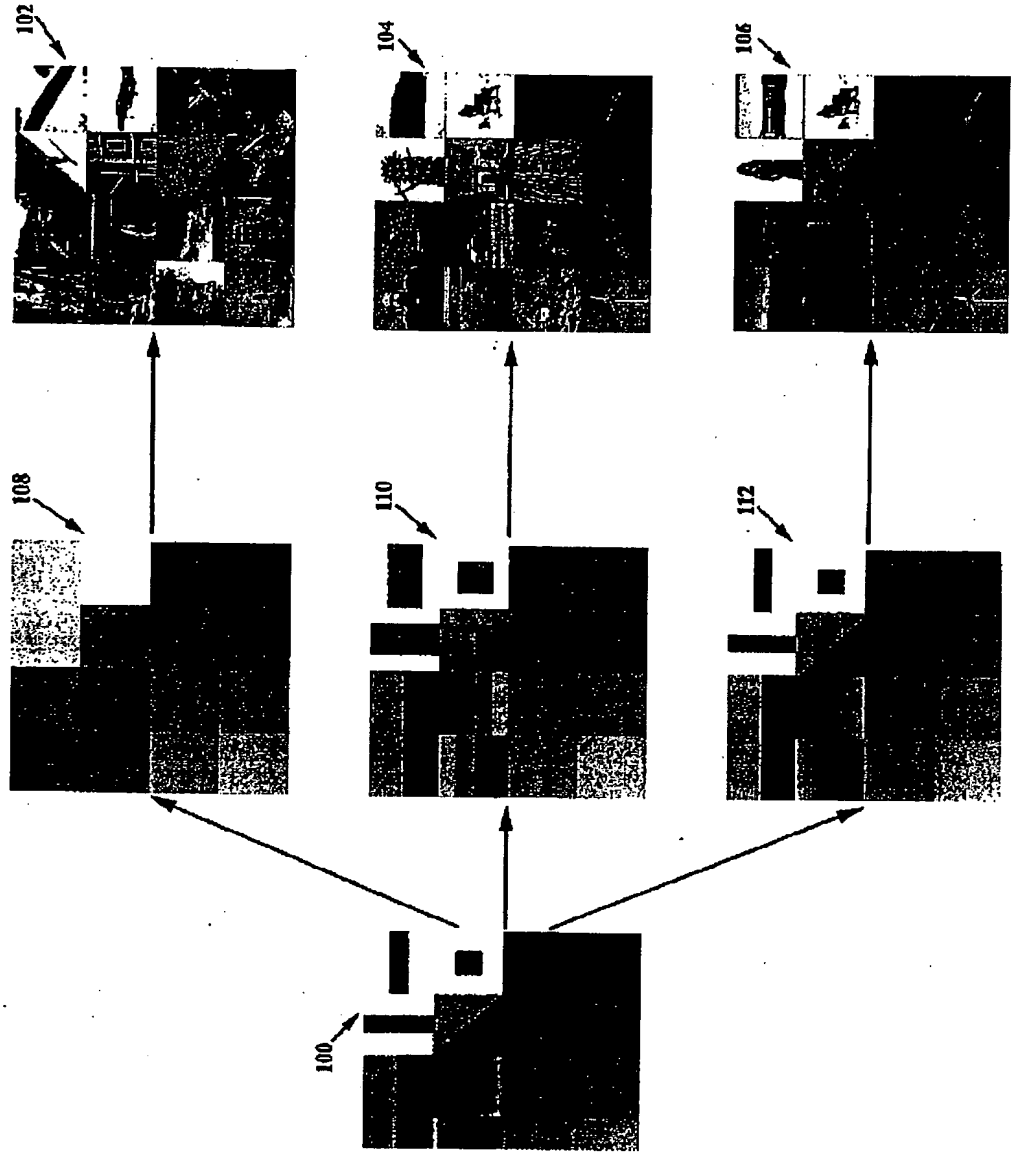
【図３】


LOAD TARGET IMAGES
60
LOAD SOURCE IMAGES
62
SPECIFY TILE ARRAY
64
ALL TILES ?
88
Y
N
WRITE LIST TO TEXT FILE
90
PROGRAM READS LIST AND ASSEMBLES BITMAP
92
GET NEW TILE
68
GET NEW TARGET IMAGE SUB-REGION
70
GET NEW SOURCE IMAGE PIXEL
72
COMPUTE RGB RMS ERROR AND SUM
74
MORE SUB-REGIONS ?
76
IS SUM LOWEST SO FAR ?
78
RECORD INDEX AND ERROR
80
MORE SOURCE IMAGES ?
82
GET NEW SOURE IMAGE
84
MARK LOWEST SUM AS USED AND ALLOCATE
86

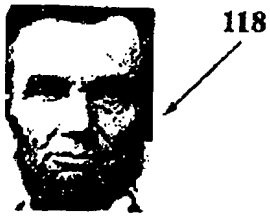
【図４】


Target image
with 8x12 tiles
14
22
22 TILES
66
Single tile with 16x16 sub-regions
66 66 66
sub-regions
22
TILE

【図５】



【図６】



1. Abstract

A mosaic image is formed from a database of source images. More particularly, the source images are analyzed, selected and organized to produce the mosaic image. A target image is divided into tile regions, each of which is compared with individual source image portions to determine the best available matching source image by computing red, green and blue channel root-mean square error. The mosaic image is formed by positioning the respective best-matching source images at the respective tile regions.

2. Representative Drawing

Figure 1